SAM DA SILVA DEVINCENZI

UM MODELO DE SUPORTE DE QOS PARA APLICAÇÕES DE TEMPO REAL

FLORIANÓPOLIS – SC
2004

UNIVERSIDADE FEDERAL DE SANTA CATARINA

PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

UM MODELO DE SUPORTE DE QOS PARA APLICAÇÕES DE TEMPO REAL

Dissertação submetida à Universidade Federal de Santa Catarina como parte dos requisitos para a obtenção do grau de Mestre em Engenharia Elétrica

SAM DA SILVA DEVINCENZI

Florianópolis, abril de 2004

Agradecimentos

Primeiramente agradeço a toda minha família, em especial ao meu pai e a minha mãe que tornaram este trabalho possível me sustentando e me apoiando em Florianópolis durante o tempo necessário. Agradeço também a minha namorada Paula, que aguentou minhas constantes trocas de humor durante toda esta fase, e sua família. Não posso deixar de agradecer também o apoio dado pelas seguintes pessoas, meu irmão “Mano” e meus amigos Éder, Maurício e Floriano que não deixaram eu desistir nas horas difíceis.

Outros agradecimentos são destinados aos professores Joni e Frank, por não terem me abandonado nunca, e por terem possíbilitado a realização deste projeto real.

Agradeço também a DEUS a quem me apeguei nos momentos difíceis (que não foram poucos !!) e que nunca me faltou com sua presença.

Resumo da Dissertação apresentada à UFSC como parte dos requisitos necessários para a obtenção do grau de Mestre em Engenharia Elétrica.

UM MODELO DE SUPORTE DE QOS PARA APLICAÇÕES DE TEMPO REAL

Sam da Silva Devincenzi

Abril/2004

Orientador: Joni da Silva Fraga, Doutor.
Co-orientador: Frank Siqueira, Doutor.
Área de concentração: Sistemas de Tempo Real
Palavras-chave: Sistemas de tempo real, CORBA, QoS, adaptação de *deadline*
Número de Páginas: 79

Este trabalho apresenta o desenvolvimento de um modelo computacional para possibilitar que a arquitetura SALE possa atender as necessidades de tempo real das empresas virtuais. O modelo foi desenvolvido com a utilização das especificações do RT-CORBA (uma extensão do padrão CORBA para tempo real) para a definição de alguns componentes de sua estrutura, assim como a utilização de especificações de QoS para capturar as necessidades do usuário. Incluído na estrutura do modelo, foi desenvolvida uma heurística para adaptação de deadlines perdidos, que é baseada na diminuição do deadline de uma tarefa por um coeficiente de ajuste a fim de aumentar a prioridade desta, fazendo desta forma, com que a freqüência de escalonamento das tarefas seja elevada. Como validação da proposta foram feitas simulações de uso do modelo em um simulador já existente, que foi adaptado para o correto uso, onde um número de tarefas periódicas eram preestabelecidas e tentavam executar respeitando seus deadlines em um servidor. Os resultados obtidos com estas simulações mostraram que a heurística de adaptação de deadline proposta no modelo, quando analisa em seu desempenho individual de atuação sobre uma tarefa específica, atinge resultados satisfatórios, visto que superou a abordagem utilizada para comparação (escalonamento *EDF*) com rendimentos mais expressivos.

Abstract of Dissertation presented to UFSC as a partial fulfillment of the requirements for the degree of Master in Electrical Engineering.

A QOS SUPORT MODEL TO REAL-TIME APPLICATIONS

Sam da Silva Devincenzi

April/2004

Advisor: Joni da Silva Fraga, Doctor.
Co- Advisor: Frank Siqueira, Doctor.
Area of Concentration: Real Time Systems
Keywords: Real Time Systems, CORBA, QoS, *deadline* adaptation
Number of Pages: 79

This work presents the development of a computational model to allow the SALE architecture to attend the real-time needs of virtual companies. The model developed used the RT-CORBA specifications (an extension of CORBA pattern for real-time) for the definition of some structural components, like the utilization of QoS specifications to capture the user needs. Included in the model structure, a heuristic to lost deadlines adaptation, that is based on the reduction of the deadline of a task by a tuning coefficient has been developed in order to increase the priority of this task. By doing so the scheduling frequency of the tasks gets high. As validation of the proposition have been done simulations of the model in use in an existent simulator, adapted to the right use, where a number of periodical tests were pre-established and tried to perform respecting its deadlines in a server. The results obtained from these simulations have shown that when analyzed in its individual performance of action over specific task, the deadline adaptation heuristics presented in the model reaches satisfactory results, seeing that it has overcome the approach used to compare (*EDF* scheduling) with a more expressive performance.

Sumário

Lista de figuras

Lista de gráficos

Lista de tabelas

1. Introdução

1.1 Motivação

Com a evolução das plataformas computacionais e de comunicação, passou a se popularizar entre os usuários de computadores uma série de aplicações que não aceita o modo como serviços são oferecidos pelos sistemas tradicionais disponíveis nos dias de hoje, que são baseados na política de melhor esforço (*best-effort*). Estas aplicações impõem requisitos de qualidade sobre os serviços por elas utilizados. Aplicações com tais características podem ser encontradas nas mais diversas áreas, englobando desde sistemas multimídia e ferramentas de trabalho cooperativo até sistemas de controle.

De maneira semelhante, uma série de sistemas corporativos, como os sistemas de manufatura, sistemas bancários e plataformas de telecomunicações, devido às suas exigências de segurança, confiabilidade e correção temporal, vêm utilizando hardware e software com características especiais que permitem o atendimento de requisitos de qualidade de serviço. No entanto, estes sistemas estão migrando para plataformas comerciais abertas como forma de reduzir custos, simplificar a manutenção e facilitar a interoperabilidade entre sistemas. Deste modo, é necessário que estas plataformas passem a levar em consideração os requisitos de qualidade das diversas aplicações em execução.

Os requisitos de qualidade exigidos pelas aplicações podem ser relacionados a diversos aspectos de qualidade, como por exemplo:

- Requisitos Temporais: algumas aplicações exigem garantias de que recursos e serviços serão fornecidos de modo determinista – ou seja, obedecendo a prazos delimitados – ao invés da forma não-determinista como se comporta a maioria dos sistemas operacionais e plataformas de comunicação atuais.

- Requisitos de Confiabilidade: por determinação dos usuários, certos serviços e aplicações podem ter que funcionar ininterruptamente ou apresentar tempos de parada extremamente exíguos.

- Requisitos de Segurança: de modo a proteger o acesso indiscriminado a dados e serviços, se torna preciso garantir altos níveis de segurança nos sistemas computacionais e de comunicação, limitando o acesso ao sistema a usuários autorizados e também restringindo as operações que podem ser executadas por estes usuários.

Aspectos qualitativos como os descritos acima podem ser especificados estaticamente (durante a construção do sistema) ou dinamicamente (em tempo de execução) na forma de requisitos de qualidade de serviço (QoS). Técnicas de especificação e obtenção de QoS, inicialmente adotadas para descrever os requisitos de aplicação impostos sobre o desempenho do serviço de rede, vem sendo usadas para descrever todos os requisitos de aplicações, tanto em relação à sua execução na máquina hospedeira quanto à sua comunicação fim-a-fim via rede de computadores.

As aplicações citadas anteriormente sofrem de uma outra limitação, que muitas vezes dificulta o seu desenvolvimento e limita a qualidade do produto final: a dificuldade que existe ao lidar com os aspectos não-funcionais da aplicação. Aspectos não-funcionais, como a comunicação na rede, a diversidade de protocolos, os diferentes formatos de dados, a interação com serviços distribuídos, etc. devem ser tratados de maneira transparente para a aplicação. Este problema vem se somar à necessidade de prover corretamente a funcionalidade básica da aplicação, ou seja, oferecer algum tipo de serviço ao usuário. Várias técnicas foram propostas para tentar resolver este problema, como o uso de módulos e bibliotecas de software, arquiteturas cliente-servidor ou *multi-tier*, *frameworks*, protocolos de meta-objetos, etc. No entanto, nenhuma das soluções propostas anteriormente conseguiu facilitar o desenvolvimento de software e ter uma aceitação tão grande e em tão pouco tempo quanto as plataformas de *middleware* para computação distribuída. Através das plataformas de *middleware*, os programadores podem facilmente desenvolver aplicações distribuídas sem se importar com os requisitos não-funcionais das

7. Referências

[1] Steve Vinoski. “CORBA: Integrating Diverse Applications Within Distributed Heterogeneous Environments ”. Revista IEEE, vol 35, nº 2, Fevereiro de 1997.

[2] Douglas C. Schmidt. “Developing Distributed Object Computing Applications With CORBA ”. http://www.cs.wustl.edu/~schmidt/PDF/corba4.pdf

[3] Frank Siqueira. “ CORBA – Common Object Request Broker Architecture “. Em nota técnica do Laboratório de Controle e Micro-Informática da Universidade Federal de Santa Catarina. Florianópolis, SC, Brasil.

[4] Rafael R. Obelheiro. “*Modelos de Segurança Baseados em Papéis para Sistemas de Larga Escala: A Proposta RBAC-JaCoWeb* ”. Dissertação de mestrado, Programa de pós-graduação em Engenharia Elétrica, Universidade Federal de Santa Catarina, Florianópolis, SC, fevereiro de 2001.

[5] Disponível em:
http://www.omg.org/technology/documents/formal/naming_service.htm

[6] Disponível em:
http://www.omg.org/technology/documents/formal/transaction_service.htm

[7] Disponível em :
http://www.omg.org/technology/documents/formal/event_service.htm

[8] Disponível em :
http://www.omg.org/technology/documents/formal/c++.htm

[9] Disponível em :
http://www.omg.org/technology/documents/formal/corba_iiop.htm

[10] Joni Fraga, Lau Cheuk, Carla Merkle e Carlos Montez. "*Suporte para Aplicações Críticas nas Especificações CORBA: Tolerância a Faltas, Segurança e Tempo Real"*, 19° Simpósio Brasileiro de Redes de Computadores, 21 a 25 de maio de 2001, Florianópolis, Brasil.

[11] Especificações RT-CORBA versão 1.1, agosto de 2002. http://www.omg.org

[12] Douglas Schmidt e Fred Kuhns. "An *Overview of the Real-time CORBA Specification"*. Revista IEEE junho de 2000.

[13] Douglas Schmidt. " TAO: A Pattern-Oriented Object Broker for Distributed Real-time and Embedded Systems ". http://dsonline.computer.org/middleware/articles/dsonline-TAO.html

[14] Carlos Mitidieri. " TAO – Implementação e Modelo de Programação". http://www.delet.ufrgs.br/~miti/Disciplines/CMP167/t1_teoria_tao/

[15] Douglas Schmidt. "Object Interconnections: Real-time CORBA, Part 2: Applications and Priorities". http://www.cuj.com/experts/2001/vinoski.htm

[16] Douglas Schmidt. " Techniques for Enhancing Real-time CORBA Quality of Service ". http://www.cs.wustl.edu/~schmidt/PDF/IEEE-proc.pdf

[17] Disponível em:
http://www.cs.wustl.edu/~schmidt/

[18] Douglas Schmidt, David L. Levine e Sumedh Mungee. "The Design of the TAO Real-Time Object Request Broker". Em Computer Communications, Elsivier Science, Volume 21, Numero 4, Abril, 1998.

[19] OMG - Object Management Group. http://www.omg.org

[20] Frank Siqueira e Vinny Cahill. “Quartz: A QoS Architecture for Open Systems”. In Proceedings of the 20th IEEE International Conference on Distributed Computing Systems - ICDCS'2000, Taipei, Taiwan, April 2000.

[21] Frank Siqueira. “Especificação de Requisitos de Qualidade de Serviço na Análise e Projeto de Sistemas” Documento Interno, Departamento de Informática e Estatística, Universidade Federal de Santa Catarina. Florianópolis, SC, Brasil.

[22] Cristina Aurrecoechea, Andrew Campbell e Linda Hauw. “A Survey of Quality of Service Architectures”. In 4th IFIP International Conference on Quality of Service, Paris, France, March 1996.

[23] Svend Frolund e Jari Koistinen. “Quality of Service Specification in Distributed Object Systems Design”. In proceedings of the 4th USENIX Conference on Object Technologies and Systems (COOTS ‘98), Santa Fe, New Mexico, April 27-30, 1998.

[24] Svend Frolund e Jari Koistinen. “Quality of Service aware distributed object systems”, In proceedings of the 5th USENIX Conference on Oriented Technologies and Systems (COOTS '99) San Diego, California, USA, May 3-7,1999.

[25] John Zinky, David Bakken, Richard Schantz “Architectural Support for Quality of Service for CORBA Objects”, Theory and Practice of Object Systems, Vol. 3(1), January 1997.

[26] “Introduction to Quality of Service”, White Paper da Nortel Networks www.nortelnetworks.com.

[27] “Quality of Service Solutions”, White Paper da Cisco Systems. www.cisco.com.

[28] Gonçalo Quadros, Edmundo Monteiro, Fernando Boavida. “QoS em Sistemas de Comunicação: Desafios, Aproximações e Capacidades Desejáveis”, In Proceedings da I Conferencia Nacional de Telecomunicações, Aveiro, Abril de 1997.

[29] C.B. Montez, Um Modelo de Programação e uma Abordagem de Escalonamento Adaptativo Usando RT_CORBA, Univ. Fed. de Santa Catarina, Departamento de Automação e Sistemas, Florianópolis, Brasil, 2000.

[30] A. Cervieri, Uma Abordagem de Escalonamento Adaptativo no Ambiente Real-Time CORBA, Univ. Fed. do Rio Grande do Sul, Instituto de Informática, Porto Alegre, Brasil, 2002.

[31] Jean Marie Farines, Joni S. Fraga e Rômulo De Oliveira, Sistemas de Tempo Real, p. 201, Escola de Computação 2000, São Paulo, Brasil

[32] C. Lu, J. A. Stankovic, T.F. Abdelzaher, G. Tao, S.H. Son, M. Marley. "**Performance Specifications and Metrics for Adaptive Real-Time Systems**". 21° IEEE Real-Time Systems Symposium, Nov. 2000.

[33] C. Mcelhone, A. Burns. "**Scheduling Optional Computations for Adaptive Real-Time Systems**". Journal of Systems Architectures 2000. http://www.researchindex.com.

[34] F. Siqueira. "**Quartz: A QoS Architecture for Open Systems**", Department of Computer Science, Trinity College Dublin, December 1999.

[35] J. S. Fraga, R. J. Rabelo, F. A. Siqueira, J. Farines, C. B. Montez. "**Suporte para Aplicações em Sistemas Abertos de Larga Escala: o Projeto SALE**". Dep. de Automação e Sistemas, Univ. Fed. de Santa Catarina, junho de 2001.

[36] J. Browne, P. Sackett, J. Wortmann, **"Future Manufacturing Systems: Towards the Extended Enterprise"**, Computer in Industry, Special Issue on CIM in the Extended Enterprise, Vol. 25(3), pp. 235-254, 1995.

[37] Object Managemet Group, **"Realtime CORBA 1.0: Revised Submission"**, OMG document orbos/98-12-10, Dezembro de 1998.

[38] OMG, **"Security Service: v1.2 Final"**, Object Management Group, Document 98-01-

02, in CORBAServices Nov. 1998.

[39] Object Management Group, **"Fault-Tolerant CORBA"**, Joint Revised Submission Document orbos/99-12-08, December 1999.

[40] Disponível em:
http://www.lfbs.rwth-aachen.de/users/stefan/rofes/docs/01-08-34.pdf

[41] Disponível em:
http://www.omg.org/technology/documents/vault.htm#corba_iiop

[42] C. Montez, J. Fraga, R. Oliveira. "Lidando com Sobrecarga no Escalonamento de Tarefas com Restrições Temporais: A Abordagem (p+i, k)-firm", Anais do *VIII Simpósio de Computação Tolerante a Falhas*, Campinas, Brasil, 1999.

[43] M. Handaoui, P. Ramanathan. " A Dynamic Priority Assignment Technique for Streams with Deadline (m,k)-firm", *IEEE Trans. On Computer*, April 1995.

[44] J. Y. Chung. " Scheduling Periodic Jobs that Allow Imprecise Results", *IEEE Trans. On Computer*, 1990.

[45] K. G. Shin, C. L. Meissner. "Adaptation and Graceful Degradation of Control System Performance by Task Reallocation and Period Adjustment", *11° Euromicro Conference on Real-Time Systems*, jun. 1999.

[46] R. J. Rabelo **"Interoperating Standards in Multiagent Manufacturing Scheduling Systems"**, International Journal of Computer Applications in Technology (IJCAT), 2000.

[47] Disponível em:
http://www.corba.org/standards.htm

[48] Li, G. "**Distributing Real-Time Objects : Some Early Experiences**", APM/ANSA External Paper 1231.01, Cambridge, UK, 1994.

[49] Frank Siqueira. "**Especificação de Requisitos de Qualidade de Serviço em Sistemas Abertos: A Linguagem QSL**". 20º Simpósio Brasileiro de Redes de Computadores (SBRC'2002). Búzios - RJ, Brasil, Maio de 2002.

[50] Object Management Group, **"Meta-Object Facility"**, RFP5, Documento 96-05-02,1996.

[51] Object Management Group, **"Portable Interceptor**", RFP - Request for Proposal OMG Document Orbos/ 98-05-05, Maio de 1998.

[52] C.L. Liu, J.W.Layland, "**Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment**", Journal of the ACM, Vol. 20(1), January 1973.

[53] J.S.Fraga, J.-M. Farines, C. Montez; "**Um Serviço de Tempo Global para Sistemas Distribuídos de Larga Escala**", 16° SBRC – Simpósio Brasileiro de Redes de Computadores, Rio de Janeiro-RJ, Maio de 1998.

[54] Maes, P. "Concepts and Experiments in Computational Reflection", In OOPSLA, Orlando, Florida – USA, 1987.

[55] B. Sprunt, L. Sha, J. Lehoczky. "Aperiodic Task Scheduling for Hard-Real-Time Systems". The Journal of Real-Time Systems, Vol. 1, 1989.

[56] D. L. Mills. "Internet Time Synchronization: The Network Time Protocol", IEEE Trans. on Communications 39, October 1991.

aplicações, que são tratados pelo *middleware* ou por serviços a ele associados. Em particular, devido à popularização das linguagens orientadas a objetos, diversos middleware para objetos distribuídos surgiram no mercado, tendo o CORBA (*Common Object Request Broker Architecture*) se firmado como um padrão mundial definido pela ISO [47] (especificamente a *IDL*, *TRADE* e o *MOF*). No entanto, as plataformas de *middleware* disponíveis atualmente tendem a ignorar os aspectos qualitativos das aplicações, pois em geral não possuem suporte para especificação e para obtenção de garantias de qualidade de serviço. No CORBA, em particular, existem extensões para lidar os requisitos de desempenho de aplicações, reunidas em um padrão chamado RT-CORBA [11]. No entanto, o RT-CORBA é de difícil utilização por exigir que os desenvolvedores de aplicações especifiquem os recursos e parâmetros do sistema operacional que serão usados em tempo de execução. No entanto, o desenvolvedor da aplicação em geral deseja se abstrair dos aspectos de mais baixo nível, lidando apenas com os requisitos impostos sobre a resposta do sistema. O fornecimento de mecanismos mais amigáveis para especificação de requisitos de QoS ainda é um assunto de pesquisa em aberto, e apesar dos esforços de pesquisa envidados para desenvolver este tipo de mecanismo, ainda não se encontrou uma solução definitiva que possa ser usada nas mais diversas situações.

1.2 Objetivos da dissertação

A plataforma SALE (Suporte a Sistemas Abertos de Larga Escala) [35] tem como objetivo central fornecer o suporte computacional necessário para a criação e operação de empresas virtuais. Para tal, esta plataforma busca integrar e conciliar mecanismos que forneçam diferentes garantias temporais, de confiabilidade e de segurança nas interações entre aplicações. Estes atributos de qualidade devem ser especificados na forma de requisitos de qualidade de serviço (QoS). Os serviços de suporte são especializados a partir destes requisitos de QoS, envolvendo diferentes mecanismos na plataforma de suporte e em camadas subjacentes. Os mecanismos necessários em uma especialização são tornados disponíveis a partir de uma interface única, os *objetos de QoS* .

A plataforma SALE adota o padrão CORBA com o intuito de permitir a interoperabilidade entre os objetos que formam uma aplicação distribuída. Os requisitos de

desempenho, confiabilidade e segurança da aplicação são fornecidos, fazendo uso das especificações *Real-Time CORBA* (RT-CORBA) [37], *Faul-Tolerant CORBA* (FT-CORBA) [39] e *CORBA Security* (CORBASec) [38], respectivamente.

O trabalho descrito nesta dissertação consiste em propor um modelo para a especificação e a obtenção de requisitos temporais conforme definido no projeto SALE. Este modelo foi definido com base em um estudo do suporte de tempo real dado pela arquitetura CORBA (RT-CORBA) e implementado pelo TAO [13]. Para tal, foi projetado um suporte de execução para aplicações distribuídas de larga escala que leva em conta os requisitos de qualidade de serviço destas aplicações, em particular aqueles impostos na forma de restrições temporais. Deste modo, aplicações que exigem determinados tempos de resposta em interações distribuídas podem ser executadas pelo suporte de execução de forma que se busque garantir o cumprimento destes requisitos sempre que isto se mostrar possível.

Como parte deste modelo, está sendo proposta uma heurística para adaptar dinamicamente os requisitos das aplicações, ajustando-os com o objetivo de fazer com que estes requisitos sejam atendidos mesmo em situações adversas. Desta forma, é possível se adaptar as condições de escalonamento para atender requisitos de QoS.

1.3 Metodologia

Neste trabalho é definido um modelo de execução de aplicações distribuídas com requisitos temporais tendo como base os mecanismos disponíveis através do *middleware* RT-CORBA. Com base no estudo destes mecanismos, foi proposta uma infra-estrutura para especificação e obtenção dos requisitos de QoS de aplicações distribuídas.

A heurística proposta como parte deste modelo de execução foi testada através de simulações realizadas em diversos cenários de execução, tendo demonstrado a sua utilidade na adaptação de parâmetros temporais das aplicações de modo a fazer com que estes sejam atendidos, na medida do possível, em situações adversas.

1.4 Organização do texto

O texto desta dissertação é dividido em 6 capítulos. Neste capítulo inicial foi descrito o contexto do trabalho apresentado.

O segundo capítulo apresenta alguns conceitos sobre sistemas de Tempo Real, como escalonamento por exemplo, e Qualidade de Serviço – QoS (adaptação, reserva de recursos, etc.) necessários para o desenvolvimento do modelo

O capítulo três apresenta a arquitetura CORBA, faz um retrato atual das especificações do RT-CORBA e apresenta alguns trabalhos que contêm QoS e RT-CORBA em seus contextos.

Os capítulos quatro e cinco trazem respectivamente, uma descrição geral do projeto SALE e a descrição do modelo desenvolvido neste trabalho (com a definição da heurística, e seu mapeamento no RT-CORBA), e as simulações, os resultados e a análise do modelo proposto.

Por ultimo, no capítulo seis são apresentadas algumas conclusões deste trabalho assim como propostas de trabalhos futuros.

2. Sistemas de Tempo Real e Qualidade de Serviço

Neste capítulo são apresentados dois conceitos fundamentais para o entendimento do trabalho que será posteriormente apresentado. Primeiramente, estaremos conceituando os sistemas de tempo real e descrevendo suas principais características, dando especial atenção ao escalonamento. Em seguida, mostraremos como os requisitos de qualidade de serviço necessários para um sistema, incluindo os requisitos temporais presentes em sistemas de tempo real, podem ser especificados, mapeados em recursos da plataforma de execução, reservados de modo a fazer com que os requisitos do sistema sejam obedecidos, e eventualmente adaptados caso os recursos necessários não se encontrem disponíveis.

2.1 Sistemas de tempo real

Sistemas de tempo real são aqueles que além dos requisitos comuns presentes em um sistema computacional, como a integridade dos resultados, apresentam ainda responsabilidades temporais quanto aos resultados obtidos. Nestes sistemas existem prazos a serem cumpridos que acarretam em mudanças em seu comportamento, fazendo com que o sistema além do comprometimento lógico das tarefas tenha também responsabilidade com valores temporais de suas execuções. Neste tipo de sistema, um resultado correto entregue fora do prazo especificado para ser apresentado tem o mesmo efeito de um resultado incorreto.

Segundo [30], a maior parte das aplicações de tempo real se comportam como sistemas reativos com restrições temporais. Ou seja, em cada evento ocorrido o sistema deve, dentro de um certo tempo (prazo) específico, obter um resultado correto. Para possibilitar a verificação de sua correção temporal, as atividades do sistema são associadas a restrições temporais, especificadas na forma de parâmetros que regem o comportamento temporal da atividade, como o período de ativação, o tempo máximo de execução, o intervalo mínimo entre ativações, e o prazo máximo de conclusão da tarefa (*deadline*).

2.1.1 Conceitos de sistemas de tempo real

Quando se observa que em sistemas de tempo real os requisitos temporais são o principal objetivo, é comum associar-se a este conceito apenas a idéia de maior velocidade computacional. Segundo [31], a rapidez de cálculo visa melhorar o desempenho e reduzir o tempo de resposta médio, enquanto em tempo real é observado que o importante é o atendimento dos requisitos temporais. A diminuição dos tempos de resposta não garante de forma alguma que os requisitos temporais serão atendidos.

A previsibilidade temporal é o principal requisito a ser observado em sistemas de tempo real [29], visto que é ela que em tempo de projeto (*previsibilidade determinista*) vai mostrar se as restrições temporais das tarefas poderão ser satisfeitas (garantia em tempo de projeto). Neste tipo de previsão, a *previsibilidade determinista* se utiliza, em uma análise *off-line*, das hipóteses de carga e falhas do sistema. Deste modo, através do conhecimento prévio da carga máxima e dos tempos envolvidos na execução das diversas tarefas do sistema, a disponibilidade de recursos computacionais e a verificação do cumprimento das restrições temporais impostas pelo ambiente pode ser realizada através de testes de escalonabilidade que considerem situações de pior caso. Ao garantir o funcionamento do sistema sob a maior carga esperada, temos a garantia de que as restrições temporais serão cumpridas em qualquer situação.

No entanto, em alguns sistemas não existe a possibilidade de prevermos em tempo de projeto (*off-line*) a carga à qual este será submetido e nem mesmo delimitarmos os tempos de execução de algumas atividades, como o tempo de acesso ao meio de comunicação, o tempo de transmissão de uma mensagem, ou o máximo tempo de espera pela liberação de um recurso compartilhado. Neste caso, não é possível determinar ao certo o comportamento desses sistemas. Nestes sistemas, ditos não-deterministas, torna-se necessário adotar uma forma de previsibilidade denominada *previsibilidade probabilística*, garantindo uma taxa de erro aceitável dentro da qual devem ser cumpridas as restrições temporais impostas pelo ambiente. Neste tipo de sistema vigoram políticas de melhor esforço (*best-effort*), dependentes da dinâmica do sistema, que buscam atender as restrições temporais das aplicações sem, no entanto, fornecer nenhum tipo de garantia.

Os sistemas de tempo real são geralmente classificados em dois tipos, *hard* e *soft,* mas em algumas literaturas como [29], é apresentado um terceiro tipo classificado como *firm*:

- *Hard real-time systems* – nestes sistemas, não existe benefício em sua execução após o estouro de seu *deadline* e sim um prejuízo crítico (também chamado de falha catastrófica). Neste tipo de sistemas é imprescindível que seu *deadline* seja sempre respeitado necessitando-se assim da *previsibilidade determinista*;

- *Soft real-time systems* – são os sistemas onde mesmo depois do *deadline* de uma tarefa ser ultrapassado, o término da execução da tarefa e a conseqüente obtenção de um resultado ainda podem oferecer benefícios. Estes benefícios diminuem proporcionalmente com passar do momento do estouro do *deadline*;

- *Firm real-time systems* – neste tipo de sistema, apesar do resultado de uma tarefa não trazer nenhum benefício após o esgotamento do *deadline*, a perda do deadline não acarreta em prejuízos críticos para o funcionamento do sistema.

Um suporte para tempo real deve refletir o tipo de sistema para o qual se destina. Em um suporte para sistemas *hard* deve haver uma total previsibilidade do comportamento do sistema, permitindo que todas as restrições de tempo sejam examinadas previamente através de testes de escalonamento de atividades. Já em suportes para sistemas *soft*, são toleradas algumas falhas em caso de sobrecarga, permitindo que seja adotada uma política de melhor esforço (*best-effort*) para estabelecer o fluxo de execução das atividades do sistema. Sistemas *firm* permitem que sejam adotadas políticas probabilistas, mas deve-se procurar reduzir ao máximo as taxas de erro limitando-as a valores aceitáveis mesmo sob situações altamente adversas.

2.1.2 Escalonamento em sistemas de tempo real

O escalonamento em sistemas de tempo real observa o conjunto de tarefas e a carga que este conjunto gera de modo a definir a ordem de execução das tarefas. A obtenção

destas informações pode ser feita de duas formas: uma quando as informações sobre as tarefas e suas restrições temporais são conhecidas em tempo de projeto, na qual a carga é classificada como **estática** e **limitada**; a outra é quando dados como tempo de chegada, execução, etc., não são conhecidos previamente, sendo que neste caso a carga é classificada como sendo **dinâmica** e **ilimitada**.

As tarefas em sistemas de tempo real podem ser:

- *Tarefas periódicas* – tarefas que são ativadas de forma síncrona, com o intervalo entre as ativações sendo o período estipulado;

- *Tarefas aperiódicas* – as quais são ativadas de forma assíncrona por eventos aleatórios.

No modelo de carga limitada e estática, as tarefas usadas para a modelagem são do tipo periódico, enquanto para modelar a carga dinâmica e ilimitada admite-se tarefas aperiódicas.

O escalonamento, propriamente dito, é o processo de ordenar o grupo de tarefas na fila de pronto para execução no processador. Estas tarefas passam geralmente por uma análise para serem definidas as possibilidades de execução. Esta análise é denominada de teste da escalonabilidade, e é ele quem define se os conjuntos de tarefas são ou não escalonáveis a partir de suas restrições temporais.

Existem diferentes políticas de escalonamento com abordagens que apresentam diferenças significativas quanto a aspectos como previsibilidade, garantia e utilização de recursos. A seguir são mostradas 3 destas abordagens:

- *Executivo cíclico* – nesta abordagem, o teste de escalonabilidade e o trabalho de escalonamento são realizados em tempo de projeto. Uma grade é organizada a partir desta análise e define qual tarefa executará e quando;

- *Garantia dinâmica* – nesta abordagem, quando uma nova tarefa chega no sistema, é realizado um teste de aceitação para ver se o novo conjunto formado pelas tarefas já existentes na fila de pronto e a tarefa que chegou, são escalonáveis. Se este novo conjunto não for escalonável, a tarefa será rejeitada;

- *Best effort* – nesta abordagem de escalonamento não há garantias, em tempo de projeto, de que os requisitos temporais serão cumpridos. O sistema executa tentando cumprir todos os prazos, mas em certos momentos isto acarreta uma sobrecarga que deve ser tratada (na tentativa de uma minimização) através de descarte de tarefas (por exemplo).

Em sistemas distribuídos, usualmente são adotados mecanismos de solução do escalonamento em etapas, segundo os quais primeiramente é escolhido o processador no qual uma determinada tarefa deve ser executada. Em seguida, ocorre o escalonamento a nível local, levando em conta a carga alocada para o nodo na primeira etapa do escalonamento.

2.1.3 Tempo real em sistemas abertos

O tratamento de questões presentes em sistemas tempo real no contexto de sistemas abertos é dificultado devido à existência de fatores conflitantes, mas não inconciliáveis. De forma a integrar características de tempo real em sistemas distribuídos abertos, devemos considerar os seguintes aspectos [48]:

- Os tempos relativos à comunicação não são delimitados, o que implica na adoção de mecanismos probabilistas para determinação do tempo gasto na interação entre os componentes de uma aplicação distribuída;

- A carga de cada máquina nestes sistemas é dinâmica e imprevisível; com isto, não podemos impor garantias quanto à execução de nenhuma das atividades do sistema dentro do tempo desejado;

- Não existe a possibilidade de sincronização dos relógios locais dentro dos níveis de granularidade desejáveis, o que impede que um tempo global seja utilizado na verificação de restrições presentes em sistemas tempo real.

Como não existe total previsibilidade acerca do comportamento do sistema, torna-se impossível garantir que as restrições temporais sejam satisfeitas. Isto conduz à adoção de políticas de melhor esforço (*best-effort*) para o escalonamento das atividades do sistema, com o intuito de evitar que atividades sejam executadas fora de suas especificações temporais.

2.2 Qualidade de serviço - QoS

QoS é definida por [20] como sendo um conjunto de características quantitativas e qualitativas de sistemas com compromisso de tempo de resposta, desempenho, confiabilidade, qualidade de saída, entre outros, necessários para atender os requisitos de aplicações. Dentre as aplicações com requisitos de QoS podemos destacar os sistemas multimídia, de tempo real e de trabalho cooperativo.

Os componentes que respondem pelos recursos do sistema possuem características individuais, ou seja, fornecem serviços com diferentes qualidades, e com isso é possível que os requisitos das aplicações sejam atendidos através da alocação de diferentes conjuntos de recursos. Mecanismos de QoS são selecionados de acordo com a especificação de QoS fornecida pelo usuário, a disponibilidade dos recursos e políticas de gerenciamento de recursos.

No gerenciamento de recursos, os mecanismos de QoS são classificados como estáticos ou dinâmicos de acordo com sua natureza. Mecanismos estáticos interpretam os requisitos de QoS da aplicação e fazem a reserva de recursos para que estes requisitos sejam atendidos em tempo de execução. Durante este processo, pode ser necessário renegociar a qualidade de serviço caso os recursos necessários não estejam disponíveis. Já os mecanismos dinâmicos monitoram a qualidade fornecida em tempo de execução, e adaptam o funcionamento do sistema e até mesmo da aplicação caso os requisitos de QoS não estejam sendo atendidos.

Segundo [20], arquiteturas de QoS são responsáveis pela integração dos mecanismos de QoS com o sistema computacional, de modo a organizar os recursos providos pelo sistema. Os contratos de QoS de uma aplicação contêm os requisitos exigidos pela aplicação, e são utilizados pelas arquiteturas de QoS de modo a procurar prover a qualidade desejada. Os mecanismos de reserva de recursos recebem requisições de usuários para uso dos recursos providos pelo sistema – como, por exemplo, largura de banda, *buffers* de memória, processos e threads. Neste caso, deve ser feito um teste de admissão onde, se o recurso está disponível, a requisição é aceita, senão ela é rejeitada.

2.2.1 Especificações de QoS

Especificação de QoS é a maneira como as aplicações que necessitam de QoS definem seus requisitos. Por exemplo, em uma aplicação de tempo real, os requisitos que serão especificados serão requisitos temporais, onde, por exemplo, a definição de um tempo *x* para um *Deadline*, vem a ser a especificação de um requisito de QoS a ser seguido.

Na etapa de especificação o projetista preocupa-se com a captura dos requisitos de maneira a levar em conta os parâmetros usados, e a política usada para gerenciar estes requisitos, visto que, geralmente, existem diferentes níveis de abstração nos quais QoS pode ser especificada e diferentes mecanismos para configurar e suportar estas especificações de QoS, particulares a cada parte do sistema.

Segundo [21], o nível de abstração representa a forma como serão descritos os requisitos de QoS. Em geral, refere-se a requisitos em nível de aplicação quando estes requisitos são especificados de acordo com a qualidade percebida pelas aplicações. Os requisitos em nível de sistema se referem aos recursos do sistema que são necessários para que a qualidade solicitada pelas aplicações seja satisfeita. De modo geral, quanto mais clara for a noção de recursos fornecidos pela plataforma, mais distante se está do que é compreensível para a aplicação. Do ponto de vista do usuário, do projetista e do desenvolvedor da aplicação, que são responsáveis por especificar os requisitos de QoS, é desejável poder especificar seus requisitos em nível de aplicação, já que se trata de

aspectos que estão mais próximos das necessidades do programador. Como exemplo, suponha uma aplicação de monitoramento de um sistema de manufatura. Seria mais natural para o projetista dizer que precisa de uma resposta em 100ms do que ter que especificar a prioridade do sistema operacional com a qual este processo será executado.

Segundo [23], especificações de QoS podem ser usadas em diferentes situações, por exemplo, durante o desenvolvimento de sistemas, para entender as requisições de QoS de componentes individuais, que habilitam completamente o sistema a ir ao encontro das metas de QoS, ou seja, do conjunto de requisitos de QoS definidos pelo usuário/programador. Especificações de QoS também podem ser usadas para negociar dinamicamente acordos de QoS entre cliente e servidor em sistemas distribuídos.

Qualquer tentativa de definir um grupo comum de parâmetros para especificações de QoS (ou seja, informações que devem ser fornecidas pelas aplicações para expressar seus requisitos de QoS) pode acabar restringindo a definição de requisitos de QoS de maneira danosa. Desta forma, é desejável que as arquiteturas de QoS sejam flexíveis de modo a permitir o uso de diferentes formatos de parâmetros na especificação de requisitos de QoS.

Segundo [22], a especificação de QoS deve cobrir os seguintes pontos:

- *Desempenho do fluxo*, que caracteriza o desempenho do fluxo de requisições do usuário. A habilidade para garantir o tráfego completo das taxas, atraso, jitter[1] de atraso e taxas perdidas, é particularmente importante para comunicações de multimídia;

- *Sincronização do fluxo*, que caracteriza a ordem de sincronização entre os múltiplos fluxos de dados relacionados;

- *Nível de serviço*, o qual especifica a ordem de confiabilidade dos recursos requeridos. O nível de serviço guia estas requisições para serem refinadas de modo qualitativo para definir o desempenho a ser garantido;

- *Política de gerenciamento de QoS*, a qual captura a ordem de adaptação de QoS (contínua ou discreta) que o fluxo pode tolerar, e define as ações a serem tomadas em um evento de violação de um contrato de QoS;

- *Custos do serviço*, onde é especificado o preço que o usuário está disposto a pagar para o nível de serviço. O custo do serviço é um fato muito importante quando consideradas especificações de QoS.

Segundo [21], a especificação de requisitos de QoS pode ser feita tanto de maneira estática (*off-line*) quanto de maneira dinâmica (*on-line*), e os requisitos devem ser associados, de maneira individual ou em conjunto para cada instância, a serviços através das abstrações de linguagem que os implementam, podendo ser aplicados a processos, módulos, objetos, dados, procedimentos e parâmetros de procedimentos.

2.2.2 Mapeamento de QoS

Visto que os requisitos de QoS são descritos em diferentes níveis de abstração, um suporte de QoS deve oferecer um mapeamento de QoS que execute a função de tradução automática entre representações de QoS de diferentes níveis (por exemplo, traduzir os requisitos no nível de aplicação – executar uma tarefa em tempo X – em requisitos do sistema operacional – usar prioridade Y para executar a tarefa).

Segundo [21], o mapeamento entre parâmetros nem sempre é direto, ou seja, um parâmetro de aplicação pode corresponder a um parâmetro de sistema, a N parâmetros de sistema, ou pode ser agrupado com outros parâmetros e convertido em 1 ou N parâmetros. Por exemplo, a codificação do áudio transmitido em uma chamada telefônica pode utilizar recursos de transmissão e de compressão de dados, em um mapeamento de 1 para N parâmetros de QoS.

Devido à possibilidade de se tratar com diferentes parâmetros, no que diz respeito a seu formato em diferentes níveis (como por exemplo, uma especificação de QoS em nível de aplicação pode expressar requerimentos de fluxo de dados, enquanto em nível de

[1] É a medida da variação do atraso entre sucessivos pacotes, por um dado fluxo de tráfego.

sistema expressa largura de banda de pico ou média, *jitter* ou restrições de atraso ou perda), é necessário prover um suporte bastante flexível que reconheça conjuntos de parâmetros diferentes e que permita que novos parâmetros possam ser definidos pelo usuário/programador.

Em [21], nota-se que o mapeamento deve permitir a tradução dos parâmetros nos dois sentidos – do nível de aplicação para sistema e também no sentido contrário – para não só traduzir os requisitos da aplicação em recursos a serem alocados, mas também para refletir possíveis renegociações e adaptações de QoS ocorridas devido à escassez de recursos na forma de parâmetros de nível de aplicação.

2.2.3 Reserva de recursos de QoS

A finalidade da reserva de recursos de QoS é de informar antecipadamente ao sistema, as necessidades das aplicações com requisitos de QoS. Mecanismos de reserva devem suportar durante toda a execução da tarefa, ao usuário, o grupo de recursos reservado, além de receber requisições de novos usuários. Caso, durante a execução, novas requisições sejam feitas, estas estão sujeitas a testes de admissão baseados nos recursos correntes e nos níveis de garantias requisitados.

Protocolos de reserva de recursos organizam a alocação de recursos no sistema operacional e no suporte de comunicação correspondentes utilizando as especificações de QoS definidas pelos usuários. Apesar de prover estas contribuições, estes protocolos são localizados em um baixo nível de abstração, o qual não é apropriado para as aplicações.

2.2.4 Adaptação de QoS

É sabido que durante o ciclo de vida da aplicação, podem ocorrer falhas de *hardware* – como o desligamento de alguma máquina – ou de *software* – como um *deadlock* em um programa durante a sua execução – que levam à indisponibilidade de alguns recursos reservados, podendo modificar a qualidade do serviço fornecido.

Tendo em vista este problema, a idéia de adaptação de recursos se faz de maneira que os mesmos sejam garantidos durante todo o tempo de processamento da tarefa, mas suas disponibilidades possam variar, de modo que quando ocorrer algum problema com o provedor de um determinado serviço, o mapeamento seja modificado e refeito para que se consiga manter os requisitos da aplicação. Caso não se consiga uma adaptação completa, ao menos se deve prover o necessário para que a aplicação siga rodando, mesmo que com menos qualidade. Um exemplo é citado em [21], onde em um sistema de manufatura, é adotado um ciclo de produção diferente (provavelmente mais lento) devido à falha de um recurso de produção (“quebra” de um robô, por exemplo). Outro exemplo seria de uma aplicação de transmissão de áudio, onde o *jitter* é levado em conta, quando ocorre o atraso fora da média do próximo pacote a ser reproduzido, o ultimo pacote recebido tem sua reprodução repetida até que o próximo seja recebido, acarretando assim em uma pequena, mas aceitável, distorção de áudio que possibilita a manutenção da QoS. No entanto, se os descartes se tornarem sucessivos, comprometendo demais a reprodução do áudio de maneira a comprometer a QoS firmada, a execução da aplicação deve ser suspensa.

2.3 Considerações finais

Em sistemas de tempo real, requisitos temporais são impostos sobre atividades em forma de QoS com o dever de serem realizados corretamente. É desejável, portanto, um suporte para QoS que compreenda estes requisitos temporais, já que o não funcionamento correto das atividades pode resultar em falha global de um sistema. Estes suportes desejados podem ser encontrados na forma de sistemas operacionais, *middleware*, ou protocolos de comunicação que procuram garantir os requisitos temporais presentes em atividades do sistema.

Neste capítulo foram apresentados alguns conceitos básicos sobre sistemas de tempo real e qualidade de serviço. Nota-se aqui que os requisitos de tempo real são parte do grande problema de provisão de QoS. Em particular, em sistemas abertos de larga escala, como nos sistemas operacionais comerciais e na Internet, temos uma maior dificuldade para poder obter garantias de desempenho das atividades em execução devido à imprevisibilidade intrínseca destes ambientes.

Ao longo deste trabalho estaremos propondo soluções para o problema do provimento de qualidade de serviço, em particular de requisitos temporais, em sistemas abertos de larga escala. Tendo em vista a disponibilidade de suportes para desenvolvimento de aplicações nestes ambientes, que serão apresentadas no próximo capítulo, não estaremos propondo algo totalmente novo. De modo a aproveitar a base de conhecimento existente, estaremos propondo soluções compatíveis com tecnologias que já são familiares para os projetistas e desenvolvedores de software.

3. CORBA

3.1 Introdução

A OMG (*Object Management Group*) é um consórcio de mais de 800 empresas interessadas em desenvolver softwares distribuídos para ambientes heterogêneos, buscando prover teoria e prática de tecnologias de orientação a objeto em desenvolvimento de softwares. Para atingir este objetivo, a OMG cria especificações gerais e proveitosas, sobre diversos aspectos da orientação a objeto, formando padrões para sistemas abertos que aumentam a portabilidade, a interoperabilidade e a reusabilidade dos softwares desenvolvidos.

A OMG deu inicio a suas atividades com a definição das especificações CORBA (*Common Object Request Broker Architecture*) cuja primeira versão foi introduzida em 1991. Em 1996 o CORBA 2.0 era lançado determinando padrões que permitiam a interoperabilidade de diferentes implementações destas especificações[41]**.** A disseminação e a própria aceitação dos conceitos do CORBA garantiram a plena aceitação destas especificações. As especificações CORBA se tornaram um padrão ISO[47]. Do mesmo modo houve também a abertura das especificações CORBA no sentido de cobrir vários tipos de aplicação e suas necessidades. Neste sentido surgiram especificações como CORBASec [38], FT-CORBA [39] e RT-CORBA[37] que estendem as especificações originais no sentido de atenderem qualidades de serviço de certas aplicações distribuídas.

Neste capítulo fazemos uma breve introdução sobre as especificações CORBA e descrevemos as características do RT-CORBA que é objeto de nossos estudos neste trabalho.

3.2 As Especificações CORBA

Uma das iniciativas da OMG é a arquitetura OMA (*Object Management Architecture*) que é o modelo de uma infra-estrutura para objetos distribuídos sobre a qual todas as especificações da OMG estão baseadas. Esta arquitetura foi proposta objetivando

a interoperabilidade de aplicações cooperantes baseadas em objetos distribuídos em sistemas abertos, ou seja, sistemas independentes de tecnologias de fornecedores de máquinas e sistemas operacionais e, também, independente das linguagens de programação usadas na construção de aplicações distribuídas. A arquitetura OMA enfatiza a portabilidade e a reusabilidade do software. A figura 3.1 mostra a arquitetura OMA composta por:

- **Objetos de Aplicações**: são criados pelo usuário e implementam a lógica funcional da aplicação conforme seus objetivos particulares, o que faz com que suas interfaces não sejam padronizadas pela OMG;

- **ORB** (*Object Request Broker*): é o principal componente da arquitetura OMA, através do qual os objetos enviam e recebem requisições de métodos de maneira transparente em um ambiente distribuído e heterogêneo, sem mostrar detalhes de mecanismos, de protocolos de comunicação, localização de objetos e modos de implementações.

- **Serviços de Objetos**: são serviços a nível de objetos que fornecem suporte básico ao desenvolvimento de aplicações distribuídas. Atuam em conjunto com o ORB para prover a infra-estrutura necessária a aplicações distribuídas [4]. Um exemplo destes serviços é o Serviço de Nomes, que permite ao cliente localizar um objeto baseado em um nome a ele associado.

- **Facilidades Comuns**: são serviços (objetos) próprios para determinadas aplicações, implementados em alto nível, próximos das aplicações dos usuários. Segundo [4], podem ser funções de propósito geral, como suporte a agentes móveis ou correio eletrônico, ou ainda a interfaces orientadas a domínios de aplicações especificas, como a área de finanças e a área médica.

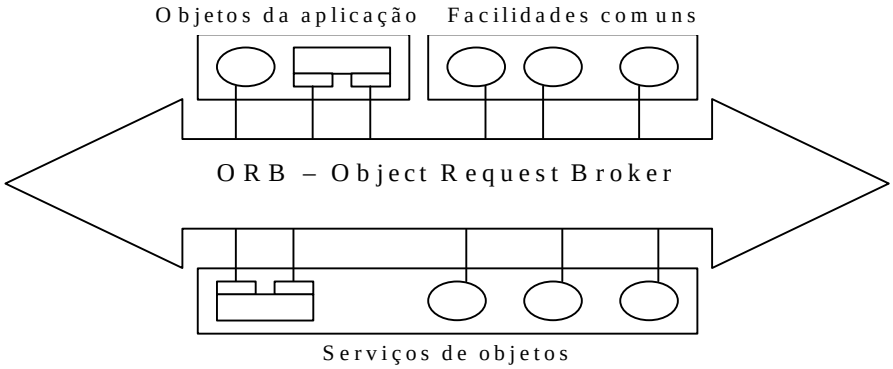

Figura 3.1 – Arquitetura OMA

3.2.1 A arquitetura CORBA

A arquitetura CORBA, mostrada na figura 3.2, foi definida pela OMG como um padrão que detalha interfaces de serviço e as características do componente ORB introduzidos no modelo OMA. Como conseqüência, a arquitetura CORBA define então objetos como unidades básicas de distribuição [4]. As interfaces CORBA implementadas por objetos em processos servidores são definidas utilizando a linguagem declarativa IDL (*Interface Definition Language*). A IDL da OMG permite a descrição padronizada de assinaturas de operações, tipos e atributos visíveis na interface de um objeto servidor. A tradução de uma interface IDL gera os *stubs* (*stubs* IDL e *Skeletons* IDL) necessários para a implementação do suporte para RMI (*Remote Method Invocation*) no CORBA.

O ORB através do seu núcleo é o elemento responsável pelo envio de mensagens GIOP (*General Inter-ORB Protocol*) em requisição/resposta concretizando as trocas necessárias em uma chamada remota. O protocolo GIOP define um conjunto reduzido de mensagens padrões que garante a interoperabilidade entre diferentes implementações de ORB. O Protocolo é dito genérico porque nas suas especificações não define um serviço de transporte especifico. A sintaxe de transferência definida no GIOP é o CDR (*Common Data Representation*) que define formatos externos para tipos primitivos e construídos. O mapeamento de mensagens GIOP sobre o serviço de transporte TCP deu origem ao Protocolo IIOP (*Internet Inter-ORB*) que tem se tornado um padrão para a Internet. Sua aceitação tem sido grande mesmo fora do contexto CORBA.

Ao receber uma requisição de um cliente solicitando a execução de um método, o Adaptador de Objeto (AO) serve como despachante localizando o *skeleton* correspondente que fará a deserialização correspondente dos valores de parâmetros da mensagem ativando o método desejado na implementação de objeto destino [3]. A operação de localização do objeto servidor envolve o uso de uma referência de objeto remoto conhecida como IOR (*Interoperable Object Reference*) no mundo CORBA.

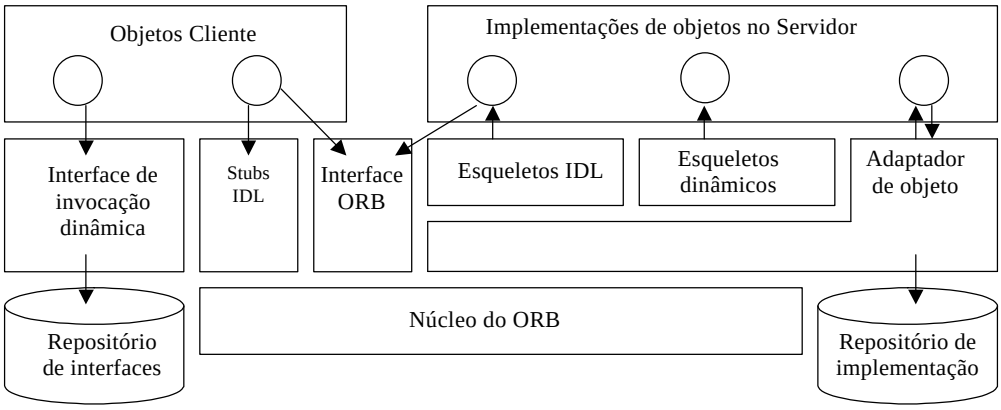

Figura 3.2 – Componentes da arquitetura CORBA

O CORBA prevê ainda interfaces dinâmicas para a invocação remota como são os casos das DIIs (Interface de Invocação Dinâmica) e dos *Skeletons* Dinâmicos. A DII é um mecanismo mais flexível que os *Stubs* IDL, porque permite a um cliente a invocação de operações em objetos sem que sejam disponível, em tempo de compilação, os códigos das *Stubs* IDL dos mesmos. Os *skeletons* dinâmicos (DSI), por sua vez, permitem que o ORB entregue requisições destinadas a uma interface desconhecida em tempo de compilação. A DSI pode ser usada para suportar novas interfaces dinamicamente, sendo empregada principalmente para construir pontes (redirecionadores de requisições) entre ORBs.

Como citado anteriormente o adaptador de objeto serve como despachante fazendo a ligação entre o ORB e a implementação do objeto CORBA. Cada interface de objeto é associada a um adaptador de objeto do mesmo tipo. O adaptador de objeto desempenha funções de ativação das implementações de objetos associadas e da geração de IORs e conversão das mesmas em *keys* que permitem a localização das implementações de objetos correspondentes no *host* servidor. A OMG introduziu o POA (*Portable Object Adapter*)

que permite ao programador desenvolver aplicações portáteis entre implementações de ORB diferentes.

Completam as funcionalidades da arquitetura CORBA os repositórios de interfaces e o de implementações (figura 3.2). O repositório de interfaces contém informações das interfaces de objetos, informações estas utilizadas pelo ORB e por clientes para descobrir as assinaturas de uma interface de um determinado objeto. O repositório de interfaces é o suporte necessário para a montagem das requisições em chamadas de métodos usando a DII. O repositório de implementações é criado com informações fornecidas de instalação de objetos, e é usado pelo ORB e pelo adaptador de objetos para localizar e ativar implementações de objetos em tempo de execução.

3.2.2 Serviços de Objetos e Facilidades

Com o intuito de complementar as funcionalidades do ORB, foram criados pela OMG os serviços e facilidades CORBA. Foram especificados pela OMG vários serviços em nível de sistema (interfaces e objetos), dentre os quais podemos citar o Serviço de Nomes, o Serviço de Eventos, o Serviço de Persistência, o Serviço de Tempo, o Serviço de Ciclo de Vida, etc. O conjunto destes serviços é denominado (Especificações de) Serviços Comuns (COSS - *Common Object Services Specificaction*s).

Assim como os serviços, as facilidades CORBA (CORBA Facilities) são utilizadas por diversas aplicações, sendo divididas em dois grupos: Facilidades horizontais e Facilidades verticais. As facilidades horizontais englobam a utilização dos serviços em várias aplicações, independente da área da aplicação, e são divididas em quatro categorias: Interface do usuário, Gerenciamento de informação, Gerenciamento de sistemas e Gerenciamento de tarefas. As facilidades verticais são utilizadas em áreas de aplicações específicas, como a área médica, de manufatura, de contabilidade e em simulações distribuídas.

3.3 RT-CORBA

Ao longo dos últimos anos, tecnologias como o CORBA vem se estendendo para domínios antes caracterizados por soluções proprietárias no que se refere a suportes de *midleware* para aplicações distribuídas. É o que acontece atualmente com as áreas de controle e automação industrial, médica, de telecomunicações, entre outras. Este processo de usar também padrões em suportes de *middleware* nestas aplicações antes fechadas é justificado pelo anseio, sempre desejável, por sistemas abertos que facilitem a possibilidade do reuso e da integração de softwares provenientes de diferentes fornecedores.

Neste sentido, com o intuito de fornecer um *framework* provendo um conjunto de interfaces padronizadas que suportassem aplicações submetidas a restrições de tempo fez com que a OMG introduzisse suas especificações RT-CORBA.

3.3.1 A arquitetura RT-CORBA

A versão 1.1 do RT-CORBA[11] determina um conjunto de extensões ao CORBA para permitir o tratamento explicito das questões de tempo real em aplicações distribuídas de tempo real. A figura 3.3 ilustra as proposições da especificação feita pela OMG. Na seqüência, é apresentada uma visão geral sobre alguns componentes da arquitetura RT-CORBA.

O RT-ORB mostrado na figura 3.3 pode ser entendido como uma extensão do ORB que fornece suporte para objetos com requisitos de tempo real. O RT-CORBA, entre outras coisas, oferece interfaces para selecionar e configurar protocolos de camadas subjacentes que sobre os quais o ORB envia suas mensagens de *request*/*reply* nas comunicações entre cliente e servidor. Estas facilidades de configuração permitem que, por exemplo, em uma aplicação de tempo real seja escolhido um protocolo determinista (com latência máxima conhecida). Na figura esta facilidade de seleção de protocolos é indicada por “outros”, numa alusão ao uso de protocolo diferente do TCP.

A versão 1.1 do RT-CORBA enfatiza escalonamentos de prioridades estáticas enquanto os escalonamentos de prioridades dinâmicas são especificados em um documento complementar para o escalonamento dinâmico[40]. Em ambos casos, as unidades de escalonamento são *threads*. Métodos requisitados são despachados para *threads* que assumem prioridades dinâmicas na execução dos mesmos.

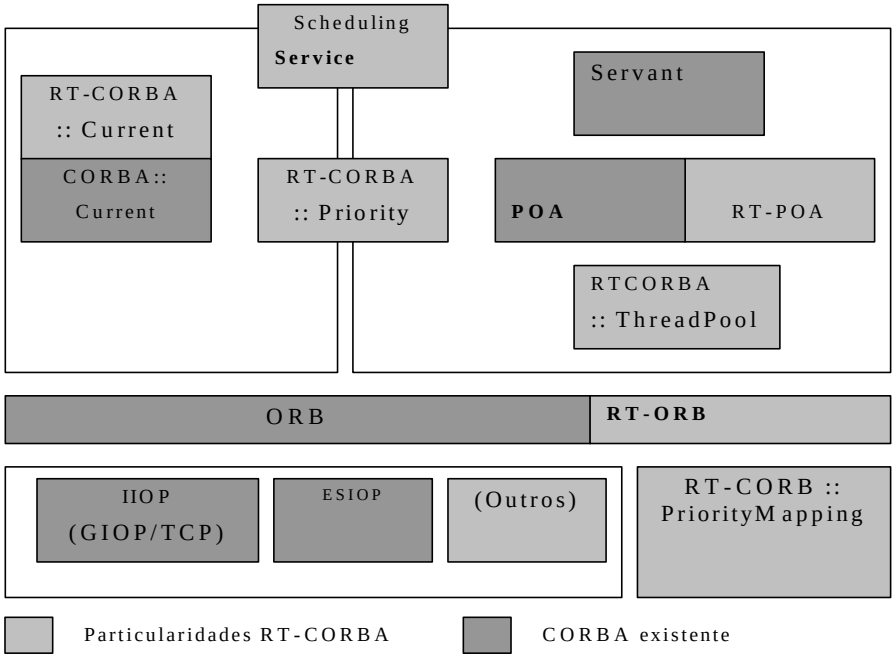

Figura 3.3. A arquitetura CORBA com extensões RT-CORBA[11]

O RT-CORBA seguindo a linha da OMG padroniza interfaces que permitem o manuseio direto das características de *threads* de tempo real. O conceito de *thread* deve ser suportado ou por bibliotecas especiais ou pelo próprio núcleo do sistema operacional de tempo real. Mesmo nas versões iniciais do RT-CORBA, já era enfatizado o uso da abstração de *pool* de *threads*, no lado do servidor, para atender os despachos do escalonador. Um *pool* pode ser criado com um número fixo de *threads* para as quais o ORB, através de um POA, vai encaminhar as requisições de clientes solicitando a execução de métodos.

Em escalonamentos de prioridades estáticas, a pré-alocação de *threads*, desde que bem dimensionada, pode ajudar a reduzir inversões de prioridades muitas vezes impostas pela fila de requisições. Se o recurso *threads* é suficiente para atender a vazão de invocações concorrentes, não haverá inversões. A fila de requisição pode também ser

atendida por uma divisão por prioridades do *pool* de *threads* na alocação estática das mesmas. Ou seja, no despacho teremos diferentes grupos de *threads* atendendo as diferentes prioridades das requisições. Neste caso, o sistema pode também oferecer alguma flexibilidade: se um grupo de *threads* do *pool* atribuído a uma prioridade está sendo sub-utilizado devido a baixa demanda de requisições neste nível de prioridades, então *threads* deste nível podem ser re-alocadas para níveis mais sobrecarregados.

Para a implementação de políticas com prioridades dinâmicas, o RT-CORBA permite que o *pool* tenha um numero de *threads* variando dinamicamente de acordo com as mudanças nas necessidades dos recursos computacionais.

As especificações RT-CORBA introduzem também uma Prioridade RT-CORBA (*RT-CORBA Priority*) definida para ter uma validade em todo o sistema distribuído, independente das implementações de ORB ou dos sistemas operacionais usados. A intenção é prover prioridades consistentes em requisições que não perdem suas validades mesmo que passem por nós com ambientes operacionais usando diferentes esquemas de prioridades. Somente valores entre 0 (*MinPriority*) e 32767 (*MaxPriority*) são válidos para o tipo *Priority* [11], onde quanto maior o número, maior a prioridade.

Valores de prioridade CORBA associados com requisições de invocação de métodos devem ser mapeados em prioridades das *threads* que atendem aos despachos correspondentes. O RT-CORBA define tipos nestes mapeamentos (*PriorityMappings*): o tipo *Native Priority* representa o esquema de prioridades que é “nativo” a um sistema operacional de um *host*. Valores de prioridades especificados em RT-CORBA *Priority* devem ser mapeados no esquema de prioridades nativo. Para permitir ao RT-ORB e às aplicações o uso destes mapeamentos, o RT-CORBA define a interface *PriorityMappings*.

A prioridade CORBA pode seguir duas abordagens de definição: *client-propagated* e *server-declared*. No modelo *client-propagated*, o servidor segue a prioridade estabelecida pelo cliente, de maneira que, quando a invocação chega ao ORB servidor, este mapeia a prioridade no seu esquema nativo, usando o *PriorityMapping*. A interface *Current* do cliente permite acesso à prioridade CORBA da invocação de método. No modelo *server-declared*, a propriedade é estabelecida no lado do servidor.

O RT-CORBA permite ainda que um cliente se comunique com um servidor por múltiplas conexões, guiando as invocações feitas com diferentes prioridades CORBA usando conexões com prioridades (*priority banded connections*).

Na dinâmica das chamadas, é possível que aplicações clientes incluam *timeouts* em suas invocações para limitar o tempo que o cliente deve ficar bloqueado uma chamada. Em relação a IDL, a extensão é encapsulada pela declaração de Módulo RT-CORBA onde todas as interfaces envolvendo objetos de tempo real são declaradas.

Na seqüência alguns dos componentes citados do RT-CORBA são descritos em mais detalhes.

3.3.2 Interface RT-ORB

A interface RT-ORB é definida como uma extensão da interface ORB do CORBA. Esta interface RT-ORB é fornecida tanto para o cliente quanto para o servidor de forma a permitir que ambos possam executar operações como o ORB_INIT para iniciar um ORB de tempo real. A interface RT-ORB também permite a clientes e servidores configurar seus objetos de tempo real como objetos CORBA.

A inicialização de um RT-ORB, como citado acima, ocorre através da operação CORBA::ORB_INIT, que ativa todas as ações necessárias para se criar um ORB de tempo real. Nesta operação, para que os valores de prioridades a serem usados possam ser controlados, dois valores são passados para a operação interna do ORB_INIT - *ORBRTpriorityrange* <*short1*>,<*short2*> - onde <*short1*> e <*short2*> representam a faixa de prioridades CORBA que as *Threads* do ORB deverão usar. <*short2*> deve ser maior que <*short1*>, caso contrario ocorre a exceção BAD_PARAM. Se o RT-ORB não puder mapear esta faixa de prioridades em seu esquema nativo de prioridades, ocorre a exceção de sistema DATA_CONVERSION e caso o RT-ORB considere a faixa de prioridades definida muito estreita para o funcionamento, ocorre a exceção de sistema INITIALIZE. A tabela 3.1 mostra a lista de exceções particulares à interface RT-ORB.

EXCEÇÕES DE SISTEMA	CÓDIGO	DESCRIÇÃO
MARSHAL	4	Tentativa de organizar objeto local.
DATA_CONVERSION	2	Falha no objeto PriorityMapping.
INITIALIZE	1	Faixa das prioridades muito restrita para o ORB.
BAD_INV_ORDER	18	Tentativa de re-associar uma prioridade.
NO_RESOURCES	2	Sem conexão para as prioridades da requisição.

Tabela 3.1. Lista de exceções do RT-CORBA

3.3.3 RT-POA

O RT-POA é uma interface derivada da interface POA do CORBA que, devido a isto, suporta além de semânticas específicas para tempo real, também as semânticas prescritas no POA. Em um ORB de tempo real, todas as instâncias devem ser feitas para o sub-tipo RTPortableServer::POA. Este sub-tipo é a declaração do RT-POA, que difere do POA em dois pontos: provê operações adicionais para suportar objetos com diferentes níveis de prioridade; e sua implementação suporta políticas de tempo real definidas em extensões de tempo real.

3.3.4 Modelo de prioridades RT-CORBA

Em sistemas de tempo real, uma das abordagens usadas para garantir restrições de tempo real é associar prioridades a mensagens, códigos e operações. No caso, visando atender as possíveis implementações de políticas de tempo real, as especificações RT-CORBA definiram um mecanismo independente de plataforma para controlar as invocações de operações. Este mecanismo, como citamos anteriormente, é a prioridade CORBA que suporta dois modelos na definição de valores de prioridades: *Client Propagated Priority Model* e *Server Declared Priority Model*.

A seleção de que modelo de prioridades que será usado é definido na interface *PriorityModelPolicy* onde, se o *Server_Declared* for o modelo selecionado, o atributo

Server_Priority indica que a prioridade seguirá o padrão do modelo CORBA gerenciado pelo POA, enquanto se o modelo escolhido for o *Client_Propagated*, o atributo *Server_Priority* indica a prioridade valida para invocações de ORB´s que não sejam para tempo real devido ao fato de um ORB CORBA convencional não possuir uma forma padrão para o cliente indicar uma prioridade relativa a invocação de métodos. A seguir um maior detalhamento dos modelos de prioridade.

Quando o modelo a ser usado é o *Client Propagated*, a prioridade CORBA é definida na invocação CORBA, e todas *Threads* criadas a partir desta invocação, devem executar em prioridade adequada definida pelo mapeamento usado. Neste modelo aplicações podem operar sobre a prioridade CORBA da invocação (estas mudanças no valor da prioridade devem ser implícitas), e as próximas invocações já serão feitas com a nova prioridade, resultante destas operações.

No modelo *Server Declared*, a prioridade CORBA é definida no contexto do servidor e deve ser colocada na referência do objeto (IOR) que retorna no *reply* correspondente da invocação de método para que a mesma possa ser conhecida pelo lado do cliente, que pode até usar esta prioridade internamente. No lado do servidor, as *Threads* que estiverem executando uma invocação, estarão com prioridade nativa obtida a partir do mapeamento de uma prioridade RT-CORBA associada a invocação, a qual foi passada pelo atributo *Server_Priority* da interface *PriorityModelPolicy* usada na sua criação.

3.3.5 Transformadores de prioridades

Os modelos descritos anteriormente, *Client Propagated* e *Server Declared*, não são suficientes através de seus mecanismos de permitir as implementações de todos os tipos usuais de políticas de escalonamento tempo real. As especificações RT-CORBA então, para tornar estes mecanismos flexíveis e adaptáveis a situações particulares, definem ainda transformadores de prioridades para que programadores possam implementar em suas aplicações modelos de prioridades mais adequados a suas necessidades. Existem dois tipos de transformadores no RT-CORBA: *InBound* e *OutBound*. O Transformador *InBound* permite que a prioridade seja alterada depois da chegada da requisição correspondente,

mas antes dela ser executada no servidor, enquanto o Transformador *OutBound* altera a prioridade no momento da saída do *reply*.

3.3.6 Pool de Threads

Com a intenção de ajudar as aplicações distribuídas que possuem uma implementação complexa do objeto, a qual é de longa duração em sua execução, o RT-CORBA especifica um modelo padrão de *pool* de *threads* como um recurso para permitir o agrupamento de *threads* por parte de um servidor para processar requisições de métodos recebidas por ele. Um *pool* de *threads* é criado com um número fixo de *threads* que um ORB utilizará para processar as requisições dos clientes de um determinado servidor.

As operações *Create_ThreadPool* e *Create_ThreadPool_With_Lane* possibilitam a criação de dois tipos diferentes de *pool* de *threads*: o com e o sem *lanes* (subgrupos de threads que assumem diferentes valores de prioridades RT-CORBA), respectivamente. A **tabela 3.2** abaixo, explicita os parâmetros que devem ser definidos nas duas operações no sentido de configurar o correspondente *pool* de *threads*.

Parâmetros de Configuração	**Modelo de *Threads* Sem Lane**	**Modelo de *Threads* Com Lane**
LANE-PRIORITY	Não se aplica.	define a prioridade de todas as *threads* que executam na *lane*.
STATIC-THREADS	define o número de *threads* que serão pré-criadas. Se o número especificado não puder ser atendido, ocorre exceção: NO-RESOURCES.	especifica o número de *threads* que serão pré-criadas (ou pré-alocadas) em uma *lane*.
DYNAMIC-THREADS	define o número de *threads* que podem ser criadas dinamicamente, depois que todas as *threads* estáticas estiverem em uso.	especifica o número de *threads* dinâmicas que podem ser criadas em uma *lane*.
DEFAULT-PRIORITY	define a prioridade CORBA para a *thread* criada de forma estática; se a criação for dinâmica, a prioridade é definida no momento em que for ativada.	Não se aplica

Tabela 3.2 – características do *pool* de *threads*

3.3.7 Binding explícito

A especificação original do CORBA suporta somente *binding* implícito, o qual define a ligação entre objetos cliente e servidor na primeira invocação de método a partir do uso da IOR do objeto no servidor pelo cliente nesta invocação. Infelizmente, *binding* implícito é inadequado para aplicações de tempo real (aplicações com requisitos de *QoS*) [12]. Tendo em vista essa dificuldade, as especificações RT_CORBA definem um *binding* explícito que permite que aplicações clientes negociem parâmetros de *QoS* apropriados a suas execuções antes de suas primeiras invocações nos objetos servidores. Estes parâmetros são tipicamente relacionados com a definição do protocolo de transporte, dos valores de *timeout* para o bloqueio do cliente na chamada (este parâmetro é usado para a detecção de falhas na comunicação ou do servidor). No lado do servidor, existem parâmetros de *QoS* que permitem a reserva antecipada de recursos necessários para a execução, tais como *pool* de *threads* e filas de requisições [10].

3.4 Trabalhos Relacionados

Dada a apresentação de alguns conceitos básicos sobre adaptação, no seguimento desta seção serão examinados alguns trabalhos existentes na literatura envolvendo CORBA, RT-CORBA e QoS.

3.4.1 TAO – THE ACE ORB

O projeto TAO (*The Ace Orb*) [13] foi criado visando suprir a necessidade de utilização de plataformas distribuídas, como o CORBA, em aplicações de tempo real ou com requisitos de qualidade de serviço (QoS), em áreas como a telemedicina, a automação da manufatura e a indústria aeroespacial. As plataformas existentes anteriormente não apresentavam nenhum tipo de suporte para sistemas distribuídos de tempo real com QoS. O projeto TAO é desenvolvido no *Center for Distributed Object Computing* da Universidade de Washington, em St. Louis – EUA. Entre as motivações do projeto TAO estão [13]:

- A definição e documentação de extensões e as otimizações necessárias para a especificação dos requisitos de QoS de aplicações de tempo real;

- A definição de modos de integração que combinem estratégias ou modelos de escalonamento de tempo real usadas nos subsistemas de I/O do sistema operacional e no ORB;

- A determinação mesmo que empírica de conceitos e estruturas necessários para habilitar RT-ORBs a garantir requisitos de QoS.

A arquitetura do TAO foi definida usando então estas motivações.

A arquitetura do TAO

Dentre as implementações de RT-CORBA existentes, o TAO se destaca, pois é um ORB que serviu de modelo para as primeiras definições da OMG principalmente no que concerne escalonamento de prioridades fixas. Mas mesmo depois nas suas atualizações sempre houve a preocupação de seguir as especificações do RT-CORBA nas suas evoluções. Outro aspecto importante é que no desenvolvimento de suas versões a procura de boas medidas de desempenho sempre nortearam este projeto.

O ORB TAO é implementado sobre o ACE [17], um *framework* para *middleware* de comunicação entre objetos distribuídos construído a partir de um conjunto de modelos desenvolvidos[14]. O ORB do TAO é construído em C++, disponível nos sistemas operacionais Linux, Solaris, Windows NT, dentre outros.

A seguir serão apresentados alguns dos componentes da arquitetura do TAO.

TAO ORB CORE

É um módulo, ilustrado na **figura 3.4**[18], desenvolvido para fornecer uma infra-estrutura eficiente em tempo de execução para comunicações uni- ou bi-direcionais, de forma síncrona ou assíncrona, entre objetos distribuídos de aplicações de tempo real. O

ORB Core permite gerenciar conexões de transporte, envio de requisições cliente e retorno das respostas (se existentes).

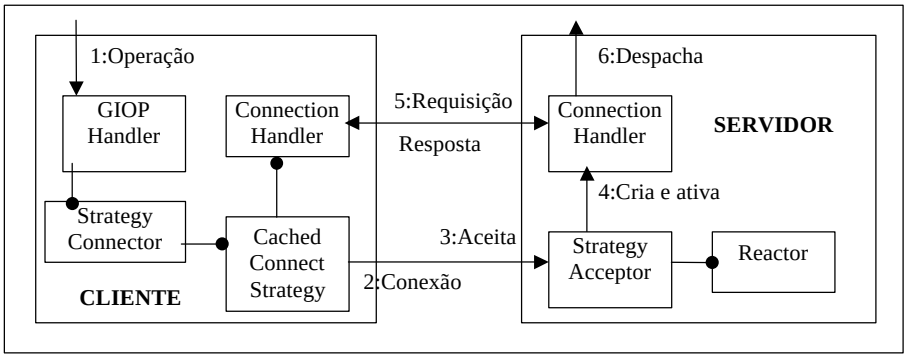

Figura 3.4 – Componentes do ORB core do TAO.

O TAO tem partes fundamentadas no *framework* ACE, o que permite características de maior flexibilidade, portabilidade e eficiência. Para simplificar as conexões com as arquiteturas concorrentes, o TAO utiliza dois modelos de estratégia com componentes ACE definidos para separar a tarefa de estabelecimento de conexão (*Conector*), dos procedimentos que ocorrem depois que esta for estabelecida (*Acceptor*). Já o modelo *Reactor* define um comportamento ACE para simplificar a demultiplexação e o despacho das conexões dos eventos destinados ao ORB core[18].

A tabela 3.3 mostra qual a função dos componentes tanto do lado do cliente como do lado do servidor no módulo principal do ORB TAO.

OPERAÇÃO	**LADO**	**FUNÇÃO**
Strategy_connector	Cliente	Armazena de forma temporária e antecipa as conexões com o servidor, salvando as configurações destas e diminuindo a latência entre a invocação cliente e a execução da operação.
Connection_handler	Cliente	Passa a requisição do cliente através do RIOP (Real-time Inter-ORB Protocol) para o *Connection_handler* do servidor.
Strategy_acceptor	Servidor	Recebe do *Reactor* a notificação de novas conexões, as aceita e as associa ao seu *Connection_handler*.
Connection_handler	Servidor	Executa em uma *Thread* as conexões passadas pelo *Strategy_acceptor*, com a prioridade de tempo-real apropriada. Também faz as chamadas ao *Object Adapter* de tempo-real para escalonar e despachar a requisição recebida do *Connection_handler* do cliente para a implementação do objeto.

Tabela 3.3 – Funções dos componentes do ORB TAO.

TAO POA

O TAO foi o primeiro ORB CORBA no qual o adaptador de objetos é implementado seguindo as especificações da OMG para *POA* e onde são usados exemplos com alta e extensiva otimização do grupo de estratégias de requisição, resultando em uma perfeita combinação para identificar objetos com referências fixas ou temporárias. Com isto, o POA do TAO desempenha com eficiência, tarefas como a demultiplexação de requisições dos clientes e o compartilhamento da capacidade de processamento do sistema entre as diferentes operações das aplicações. Portanto, estas estratégias também facilitam a obtenção de tempos limitados nos servidores no que se refere ao processamento de requisições CORBA.

Modelo de prioridade

O mapeamento de prioridades é implementado no TAO através de um objeto local chamado de *PriorityMappingManager* que se utiliza de um código localizado em uma pequena seção da aplicação para instalar o *PriorityMapping*. Este código deve ser apropriado para a versão de RT-CORBA que estiver sendo utilizada, tendo em vista a simplificação da portabilidade entre ORBs.

O TAO utiliza duas arquiteturas de prioridades RT-CORBA, em dois níveis diferentes: No nível de *threads* e no nível de portas de comunicação. No nível de *threads*, chamado de *Arquitetura Concorrente Baseada em Prioridade*, as operações podem executar os tipos de modelos de prioridades definidos na especificação RT-CORBA: o *Client Propagated* e o *Server Declared*. No segundo tipo, *Arquitetura de Conexão Baseada em Prioridade*, o modelo define que a *thread* que executara a requisição do cliente, devera manter uma conexão separada para cada prioridade de *thread* no ORB servidor, possibilitando assim aos clientes preservarem suas prioridades até o fim da operação [18]. O *Pool* de *threads* do TAO se baseia nas definições das especificações RT-CORBA referentes a *Pool* de *threads* que melhora a concorrência e otimiza o gerenciamento de memória, resultando assim na redução das inversões de prioridade devido a bloqueios em comunicações remotas.

Ligações explicitas no TAO

O TAO provê implementação de ligações explícitas (*explicit bindings*) definidas pelo RT-CORBA [16]. As conexões de *cache* do TAO são estendidas para garantir caminho para as propriedades QoS das conexões, as quais incluem prioridade de banda e atributos privados.

3.4.2 A experiência APMC

O modelo proposto em [29], chamado de APMC (*adaptative program model in* CORBA - figura 3.6), foi desenvolvido tendo em vista sistemas de tempo real. Foi a primeira experiência fazendo uso dos conceitos e suportes definido nas expecificações RT-CORBA 1.0. O APMC foi desenvolvido para atuar em ambientes de carga dinâmica e com características desconhecidas do suporte de comunicação (não são trabalhadas latências de redes ou padrões de comunicação). O modelo de tarefas incorpora uma abordagem de escalonamento adaptativo onde são assumidas tarefas periódicas não críticas, permitindo perdas de seus deadline. A qualidade do sistema ou do servidor é mantida pelo controle destas perdas para que não atingissem certos limites fora do suportável pela semântica da aplicação.

O modelo de tarefas é mapeado nos conceitos e mecanismos fornecidos pelas especificações RT-CORBA. Neste modelo são definidos objetos que concentram aspectos não-funcionais (definição de políticas) e objetos-base onde são implementados os aspectos funcionais da aplicação. Nos meta-objetos são concentrados os mecanismos para implementação da heurística usada no escalonamento adaptativo que é o (p+i, k)-*firm* [42] que faz composição da *Imprecise Computation* [44] e o (m, k)-*firm* [43]. A figura 3.5 mostra objetos base exportam suas interfaces IDL, as quais definem métodos com restrições temporais, implementados com duas versões cada: versão precisa e versão imprecisa. A seleção da versão depende das folgas para a execução dos métodos e de funções que determinam os números de perdas de deadlines de cada método de objetos no servidor.

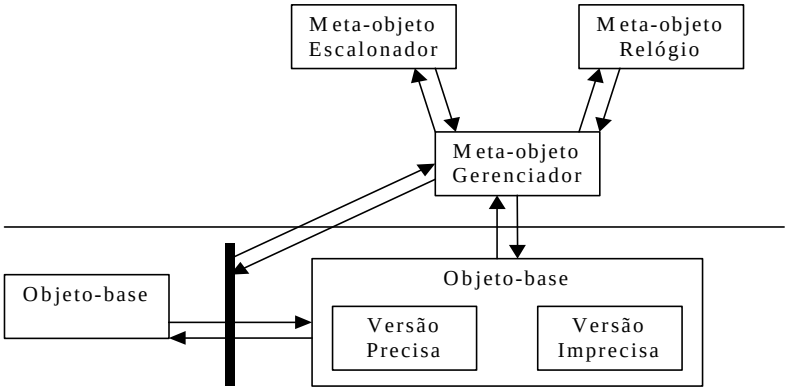

Figura 3.5 – estrutura geral do modelo APMC

Os objetos clientes neste modelo enviam requisições de métodos em objetos distribuídos de tempo real onde os requisitos de tempo real destas chamadas se restringem um deadline absoluto. Os objetos servidores recebem as invocações a seus métodos e com base no relógio local definem um deadline absoluto no momento da chegada da requisição. Objetos Meta-Objetos Gerenciadores são associados a cada objeto base no sentido de implementar a política de escalonamento adaptativo definida como (p+i, k)-*firm*. Na implementação desta política os objetos Gerenciadores interagem com o Meta-Objeto Escalonador no sentido de ordenar os pedidos de invocação de métodos segundo o escalonamento usado (através de política de prioridades). O escalonador também influi na escolha da versão precisa ou imprecisa da ativação de cada método. O Meta-Objeto Relógio é usado para definir os tempos absolutos locais no modelo.

3.4.3 Uma abordagem de escalonamento adaptativo no RT-CORBA

Em aplicações de tempo real, uma grande parte das tarefas executadas são tarefas periódicas. Uma forma de melhorar a qualidade da aplicação diante de situações de sobrecarga é atuar sobre os períodos das tarefas que se executam no servidor [30][45]. Ou seja, a sobrecarga em servidores pode ser diminuída ao aumentando os períodos de tarefas da aplicação degradando o desempenho das chamadas remotas e com isto diminuindo a carga no servidor. A degradação do sistema é então uma das formas de tratar sobrecargas.

Em [30], esta abordagem é usada com o objetivo de implementar escalonamento adaptativo sobre o RT-CORBA. Nesta proposta o autor se inspirou em [32] e em outros

trabalhos que utilizam uma realimentação para controlar variações dos períodos no sistema em função de sobrecargas (figura 3.6).

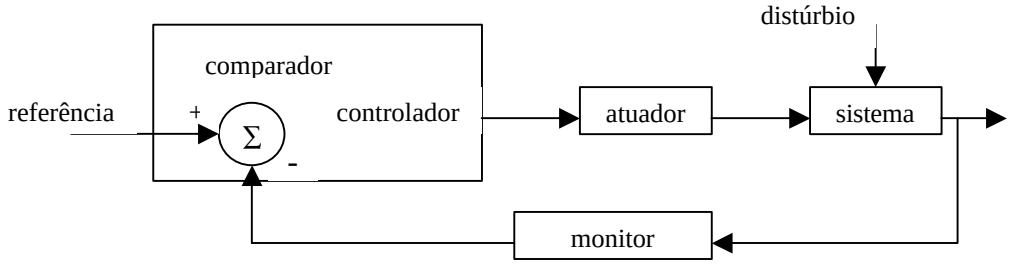

Figura 3.6 – Mecanismo básico de realimentação

A coleta das informações do sistema e a implementação do mecanismo de adaptação foram feitas de forma a afetar o mínimo possível o processamento da aplicação, pois segundo [33] o custo operacional de executar um algoritmo de adaptação no mesmo processador que as tarefas obrigatórias ou opcionais deve ser levado em consideração, de modo a procurar alternativas que sobrecarreguem menos o sistema. Nesta abordagem (como na anterior) os autores não executam retrocessos quando da perda de *deadlines* . No sentido de manter a consistência da aplicação e porque consideram que sempre haverá um ganho na qualidade da aplicação, as tarefas que perdem *deadlines* são executadas até as suas conclusões (executam os seus respectivos métodos de forma completa).

No sentido de estabelecer a heurística que controla as adaptações no sistema, foi preciso observar seu desempenho, de modo que a forma definida para quantificá-lo foi a de comparar os deadlines das tarefas com os tempos de resposta observados em uma janela de medição (DW), em um tempo t, onde a partir da definição de que o deadline é igual ao período (D=P), o desempenho do sistema (DY(t)) pode ser calculado por:

$$DY(t) = \sum_{t-DW}^{t} \frac{(Ri - Di)}{N}$$

Onde N é o numero de conclusões (de todas tarefas) na janela DW e Ri é o tempo de resposta da tarefa i em uma determinada ativação.

A partir do calculo do DY(t), um *fator* é gerado observando para isto os valores do atraso médio desejado(SDY) e uma constante proporcional(Kp), definidos pelo projetista/programador com base em cálculos matemáticos ou experimentos. O *fator* gerado é global e será utilizado para o calculo do Pefetivo (periodo efetivo) de cada tarefa, onde valores de Pmáximo (periodo máximo) e Pmínimo (periodo mínimo), especificado pelo programador/projetista, também serão utilizados.

A seguir são mostradas as fórmulas para o calculo do *fator* e do Pefetivo.

Fator = Kp DY(t) – Kp SDY

Pefetivo = Pmínimo + *Fator* (Pmáximo - Pmínimo)

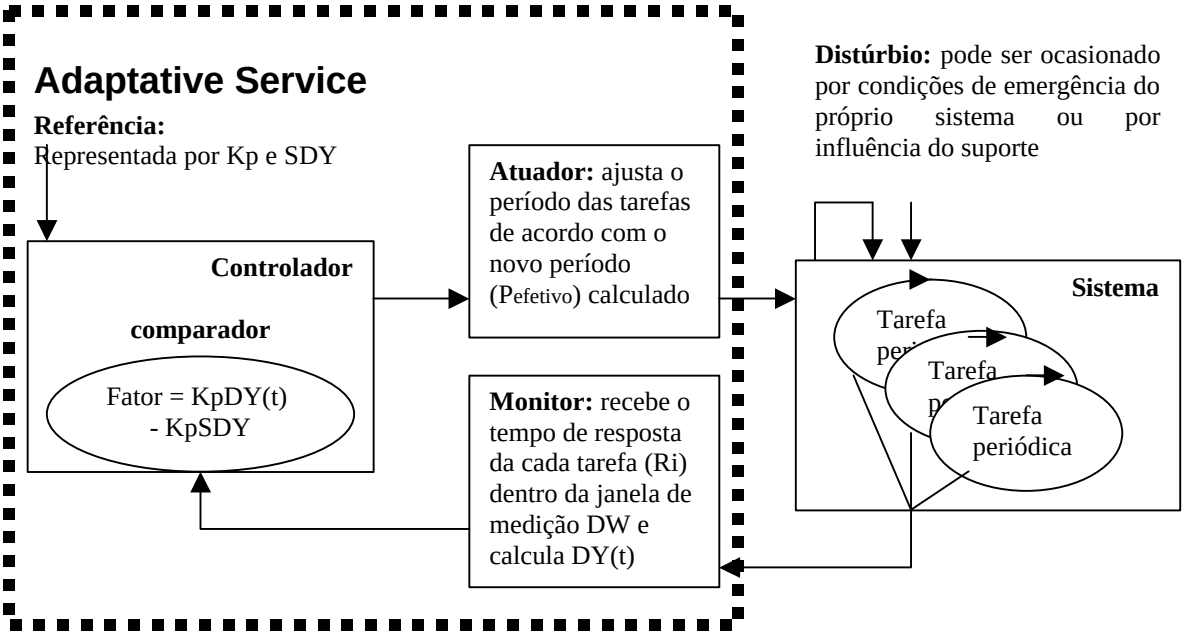

Figura 3.7 – Mecanismo de adaptação proposto [30]

Quanto ao controle de adaptação é preciso ressaltar que:

- Se Pmínimo e Pmáximo forem ajustados com mesmo valor, estará sendo definido que a tarefa não é sujeita a adaptação;

- Quanto menor o valor de Kp mais suave será a atuação no sistema.

Na figura 3.7, foi mostrado o mecanismo de adaptação proposto desenvolvido com base no mecanismo básico de realimentação.

Nesta figura pode-se notar que o *AdaptiveService* engloba as atividades de monitoração, atuação e controle.

3.4.4 Qualidade de serviço para objetos CORBA - QuO

Ultimamente, várias aplicações distribuídas têm sido desenvolvidas integrando suportes de QoS ao *middleware* CORBA, tendo em vista oferecer uma abstração de alto nível, consistente através da rede, assim como uma maior garantia nas reservas de recursos.

O modelo de uso, os requisitos de QoS e os recursos básicos devem ser integrados para uma melhor adaptação às condições de ambiente. Se considerarmos as experiências de *middleware* neste sentido, constata-se uma certa dificuldade devido à ocultação destas informações (modelo em uso, requisitos de QoS e recursos básicos) pelo uso de linguagens padronizadas de interfaces (*IDLs*). Estas interfaces padronizadas não definem nenhuma forma para reutilizar ou reconfigurar os componentes da aplicação tentando adequar as necessidades de recursos ou condições de ambiente.

A arquitetura QuO (*Qualidade de Serviço para Objetos CORBA*)[25] provê abstrações de QoS que podem ser usadas por objetos CORBA distribuídos em uma área ampla como as WAN´s. O QuO utiliza uma linguagem de descrição de QoS, chamada QDL (*QoS Description Language*), que permite que a QoS seja descrita no nível de objeto ao invés do nível de sistema. Usando QDL, os usuários descrevem:

- Um contrato de QoS com objeto servidor onde ele especifica seus requisitos de QoS relativos ao serviços providos pelo servidor;

- A estrutura interna do objeto e os elementos do sistema que devem ser monitorados e controlados para prover QoS;

- Os recursos disponíveis no sistema e seus *status*, no sentido de adaptar as variações de QoS que ocorram tempo de execução.

Regiões de QoS, que representam o *status* dos contratos de QoS para uma conexão de objeto, também podem ser descritas com QDL, permitindo as aplicações se adaptarem com a mudança das condições na troca de uma região para outra.

QuO define componentes de arquitetura que são responsáveis por tratar das garantias, medidas e adaptações de QoS na aplicação. Estes componentes de arquitetura são objetos CORBA comuns que são gerados baseando-se em descrições da IDL e da QDL. Quanto aos problemas citados anteriormente, em relação a integração de QoS com o *middleware* CORBA, o QuO resolve da seguinte maneira:

- Tornando as propriedades de sistema em entidades de primeira classe e integrando seus conhecimentos sobre tempo, espaço e origem, para então a aplicação poder ser alertada de possíveis mudanças no ambiente e mudar manualmente seu ambiente;

- Reduzindo a variação das propriedades do sistema as quais o programador deve tratar através de mascaramento;

- Expondo as decisões principais do projeto de uma implementação de objeto e como o objeto será usado, ajudado assim a reconfiguração adaptativa da aplicação;

- Suportando o reuso de vários componentes da arquitetura QuO de pontos múltiplos no ciclo de vida da aplicação.

O QuO provê um *framework* completo para desenvolver aplicações, o que reduz o esforço de programação requerido para escrever aplicações com requisitos de QoS. Mesmo assim, apresar de sua compatibilidade com o CORBA, o QuO tem seu uso em sistemas abertos limitado pelo fato de não suportar múltiplos protocolos de reserva de recursos. Em adição, sua eficiência como suporte de programação é limitada pelo fato de o modelo de

objetos do CORBA não ser apropriado para a maioria das aplicações de tempo real e multimídia.

3.5 Considerações Gerais sobre os Trabalhos Relacionados

As propostas do APMC [29] e de um controlador proporcional variando os períodos das tarefas [30] atuam basicamente no tratamento de sobrecarga de servidores na execução das invocações remotas de métodos. As duas propostas modificam parâmetros de escalonamento no sentido de atender, na melhor forma possível, a execução no servidor das solicitações recebidas. Estas propostas fazem uso do RT-CORBA na definição de seus suportes de execução distribuída e não apresentam suporte de QoS no sentido de expressar e controlar seus requisitos temporais.

O TAO embora apresente recursos para definição de requisitos de QoS, não apresenta uma abordagem de escalonamento própria para sistemas distribuídos. Reproduz apenas os mecanismos definidos pela OMG para sistemas distribuídos de tempo real. Ou seja o TAO é em essência uma implementação do RT-CORBA. Os parâmetros de QoS são especificados através de prioridades definidas no código do programa, que apresenta limitações na representação de requisitos mais elaborados.

O QuO foi desenvolvido para dar suporte de QoS a *middlewares* CORBA, tendo em vista um aproveitamento mais consistente de recursos distribuídos e atender necessidades de garantia na reserva destes recursos. O QuO apresenta uma estrutura de descrição e implementação de requisitos de QoS separada do suporte de *middleware*, ou seja, define objetos e linguagens de descrição de requisitos de QoS separados dos suportes CORBA, portanto permitindo a extensão apropriada para aplicações sem invalidar os padrões, o que invalidaria a interoperabilidade.

3.6 Considerações finais

A arquitetura CORBA (*Common Object Request Broker Architecture*) foi definida pela OMG como um padrão para o detalhamento da arquitetura OMA. Com esta finalidade, o CORBA foi desenvolvido para ser um mecanismo de software que permita a

comunicação dos objetos, independentemente de suas plataformas ou modo como foram implementados.

Tendo em vista a falta de suporte para tempo real da arquitetura CORBA, a OMG definiu as especificações RT-CORBA como um conjunto de extensões opcionais para um ORB, de modo a adicionar mecanismos de Tempo real ao seu núcleo

Neste capítulo foram detalhados conceitos sobre a arquitetura CORBA, assim como a arquitetura RT-CORBA e como esta complementa o CORBA quanto a necessidades de suporte a tempo real. Em seguida foram apresentados alguns trabalhos que envolveram CORBA e RT-CORBA em seu contexto de desenvolvimento, com o objetivo de mostrar a atualidade da área na qual este trabalho foi desenvolvido.

4. Suporte de QoS para Aplicações de Tempo Real

4.1 Introdução

Neste capítulo, é descrita a nossa proposta de suporte de *middleware* para aplicações distribuídas de tempo real. Este trabalho faz parte do Projeto SALE, que tem como principal finalidade suprir as aplicações distribuídas, em suas interações, de garantias de desempenho, confiabilidade e segurança. Os requisitos de serviço nesta plataforma são expressos através de uma linguagem própria com base na qual são gerados *Objetos de QoS* que mantêm as informações especificadas para a qualidade dos serviços desejados e atuam no sentido de especializar os serviços de suporte de *middleware*.

O trabalho descrito neste texto resume-se em mostrar como a partir da definição de requisitos de tempo real no contexto do SALE, são obtidas as restrições temporais que atuam no suporte de *middleware* no sentido de impor a qualidade de serviço desejada.

Na seqüência descrevemos sucintamente o projeto SALE como um suporte baseado em padrões de sistemas abertos, destinado a ambientes de larga escala. A caracterização de sistemas abertos é fundamentada nas especificações CORBA usando suas extensões de Tempo Real, Tolerância a falhas e Segurança para atender as necessidades de empresas virtuais. Em seguida, estabelecendo o modelo de QoS sobre a plataforma de *middleware* usada (RT-CORBA). Neste modelo, o cliente estabelece os seus requisitos de QoS e, a partir destes, são determinadas as restrições temporais que limitam as execuções no ambiente distribuído. Para a caracterização deste modelo, definimos uma heurística de mapeamento dos requisitos de QoS em parâmetros que atuam no escalonamento de tempo real das aplicações distribuídas. Esta heurística adapta os parâmetros na dificuldade de alcançar a qualidade de serviço requisitada. O modelo é mapeado nos conceitos do RT-CORBA definindo uma abordagem de escalonamento melhor esforço.

Convém salientar que não foi preocupação neste trabalho o tratamento de sobrecarga no servidor, mas sim ajustar parâmetros que definem o escalonamento no sentido de atender a qualidade de serviço contratada por um cliente.

4.2 O projeto SALE

O projeto SALE teve como objetivo inicial desenvolver uma plataforma que disponibilizasse serviços com diferentes tipos de garantias como de tempo real, tolerância a falhas e de segurança. Considerando ainda o panorama de sistemas complexos, envolvendo diferentes tecnologias, este suporte deve se apresentar mais flexível e adaptável diante das necessidades da aplicação. Segundo [35], para que isto seja atendido será necessário:

- Fazer uso extensivo de padrões abertos, que disponibilize um significante leque de tecnologias, permitindo soluções mais adaptadas à complexidade e à heterogeneidade de sistemas nestas aplicações normalmente definidas como de larga escala;

- Fornecer um ambiente conveniente e seguro para a especialização de serviços de suporte.

A arquitetura SALE é baseada no padrão CORBA que tem como objetivos prover em ambientes heterogêneos e distribuídos, a interoperabilidade, a portabilidade do código e o reuso do software. A plataforma SALE procura então fornecer para aplicações, em suas interações, garantias de desempenho, confiabilidade e segurança. Em sua arquitetura (figura 4.1), a plataforma SALE apresenta a interface Objetos de QoS que tem por objetivo receber especificações correspondentes aos atributos desejados que, então, são usados na especialização de serviços de suporte, envolvendo diferentes mecanismos de suporte a nível do *middleware* e de camadas adjacentes.

A plataforma SALE faz uso das especificações *RT-CORBA* [37], *FT-CORBA* [39] e *CORBASec* [38] para fornecer respectivamente desempenho, confiabilidade e segurança para as aplicações. Portanto, os serviços são propostos nesta plataforma tomando como base os *CORBAServices* e extensões do ORB que foram definidos nestas especificações. Adicionalmente, dependendo dos requisitos especificados pela aplicação, suportes adjacentes de comunicação em tempo real, de grupo ou segurança são usados.

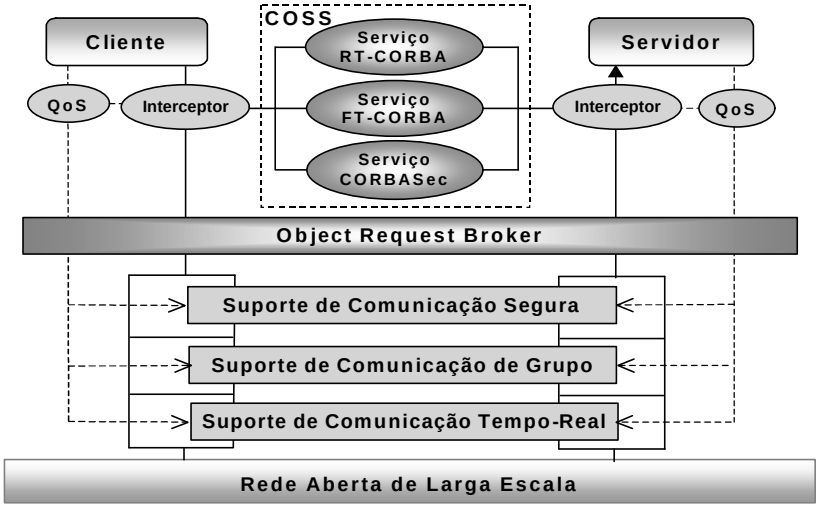

Figura 4.1 – A Arquitetura SALE

As necessidades de especialização e flexibilidade para aplicações em sistemas complexos podem ser supridas por conceitos como a reflexão computacional, na forma de protocolos Meta-Objetos [50]. A separação entre funcionalidades de uma aplicação e os controles que atuam sobre a mesma, enfatizada pelo conceito de reflexão, facilita as modificações de políticas e mesmo de comportamento de uma aplicação.

Suportes de *middleware* podem fornecer mecanismos no sentido de implementar controles reflexivos. A reflexão computacional, é alvo de padronização na OMG, mas essa padronização envolve somente mecanismos de reflexão estrutural [54]**.** A reflexão comportamental que interessa neste trabalho pode ser conseguida com o uso da noção de *interceptors* – mecanismos também padronizados nas especificações CORBA [51]. A interceptação em cada invocação de método através do ORB torna transparente a execução de controles sobre os métodos de aplicação.

O suporte proposto, que é ilustrado na figura 4.1, tem como objetivo central integrar e conciliar mecanismos que fornecem diferentes garantias de desempenho, confiabilidade e segurança nas interações das aplicações. Estes atributos de qualidade envolvendo clientes e servidores de aplicação devem ser especificados na forma de requisitos de QoS, envolvendo diferentes mecanismos no suporte a ser desenvolvido e em camadas subjacentes. Os mecanismos serão disponibilizados à aplicação a partir de uma

interface única, permitindo a especificação e o fornecimento dos diferentes requisitos de qualidade de serviço (QoS) de forma uniforme.

Aplicações sem requisitos de QoS devem fornecer serviços convencionais a partir de mecanismos usuais existentes em *middlewares* e em tecnologias de redes. Nas aplicações com requisitos de desempenho, confiabilidade e de segurança, as chamadas de objetos são interceptadas e tratadas de forma diferenciada, influenciando tanto a forma como o sistema local executa a aplicação quanto o modo como a comunicação da aplicação é tratada pelo suporte de comunicação.

4.3 O modelo computacional de suporte de QoS para tempo real

Nos capítulos anteriores foram apresentados os padrões CORBA e sua extensão *RT-CORBA*, além de conceitos sobre a utilização de mecanismos para o tratamento de requisitos de QoS. Este item tem por objetivo apresentar um modelo computacional que a partir de requisitos de QoS define as restrições e o suporte para aplicações de tempo real.

Os valores dos parâmetros de QoS definem, a partir do cliente, as restrições de tempo que envolvem a execução remota de método. Estas restrições são obtidas estaticamente a partir da especificação em uma linguagem própria para especificação de QoS, como por exemplo a *QSL* (*Quality Specification/Scripting Language*), ou em tempo de execução, caso os valores sejam alterados através do objeto QoS do lado do cliente.

O modelo da figura 4.2, com objetos de serviço e extensões do *RT-CORBA*, é uma especialização do apresentado na seção anterior. Nesta especialização, um objeto de QoS ligado ao suporte de invocação de um método remoto serve de repositório dos requisitos do cliente em relação a estas invocações e também, das políticas de mapeamento dos mesmos em parâmetros ou restrições de tempo real que atuam nestas invocações. O objeto QoS é ativado de forma transparente através do uso de interceptação[2]. No retorno da chamada, o interceptador, além de passar ao cliente a resposta de sua invocação, coleta informações e exceções sobre a execução atual para serem processadas no objeto de QoS.

[2] Um *interceptor* é um objeto que tem suas operações chamadas pelo ORB em diferentes momentos de invocação de método remoto.

Neste trabalho nós nos fixamos somente na atribuição por parte de um usuário de um *timeout* para suas requisições a um determinado método. Os seus requisitos de QoS, no caso o *timeout* para invocações de um método, serão mantidos no objeto de QoS do cliente e consultados nas interceptações que sofrem as invocações para determinar, por exemplo, o *Deadline* relativo correspondente a este método. Uma heurística é usada no cálculo do *Deadline* relativo de uma requisição que só então irá ser enviada ao servidor.

No modelo, os requisitos de desempenho do cliente são implementados, usando *prioridades* CORBA para transferir para o servidor os valores de *deadlines relativos* definidos no objeto de QoS do lado do cliente. A prioridade *CORBA* é um tipo de grandeza global que pode atravessar diferentes ORBs, dentro de mensagens de invocação. Dependendo da abordagem de escalonamento, essa prioridade pode ser usada pelos servidores para ordenar as requisições recebidas. A prioridade CORBA é usada segundo a abordagem *client-propagated*. A escolha deste estilo se deve ao mesmo admitir operar nos valores de prioridade durante o processo de invocação remota.

Durante um *binding* explícito, são vários os passos tomados para interconectar objetos, tais como localizar o objeto destino e iniciar estruturas de dados usadas na comunicação entre os objetos cliente e servidor. No lado do cliente, o *binding* pode ser usado, por exemplo, para a seleção de um protocolo de transporte apropriado. No lado do servidor, o *binding* permite a alocação dos recursos necessários na execução das requisições, tais como *threads* e *pool* de *threads* (para filas de requisição). Estes últimos são definidos como requisitos de QoS no objeto de QoS no servidor.

A proposta, por envolver aplicações na Internet, exclui protocolos deterministas que não podem ser configurados como suporte de transporte do RT-CORBA. Neste sentido, as nossas soluções de escalonamento tempo real envolvem *best-effort* e *soft real-time* devido às dificuldades de se conseguir tempos de latência limitados para a comunicação remota.

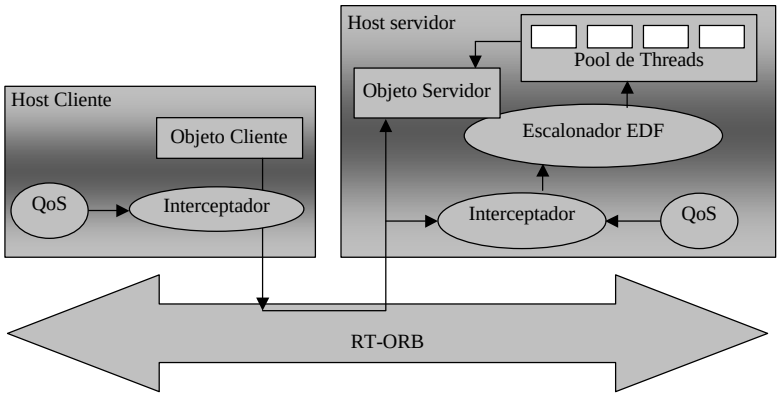

Figura 4.2: Modelo para mapeamento de requisitos de QoS

A figura 4.2 completa o cenário do modelo ainda com objetos relacionados à abordagem de escalonamento. O escalonamento no servidor é implementado em modo refletido por um objeto de serviço – o *Serviço* de *Escalonamento*. As invocações de método são ordenadas a partir de uma interceptação que aciona o serviço de escalonamento que implementa a política de escalonamento na ordenação das requisições de método. No caso é usado o *EDF* (“*Earliest Deadline First”* [52]) como política na atribuição de *lanes* (e, por conseqüência, das *threads* associadas) às requisições recebidas no servidor. Estas *lanes* ou filas de requisições estão associadas a diferentes prioridades locais. As requisições de clientes são ordenadas nestas *lanes* segundo ordem de chegada.

Interceptadores, agindo no lado do servidor, antes e depois do processamento de uma invocação, são habilitados a transformar prioridades CORBA de invocações de métodos. Após a requisição ser recebida pelo interceptador, o *deadline* absoluto da requisição é calculado, usando o relógio local, a partir do *deadline* relativo estabelecido para a chamada. A prioridade CORBA é modificada para este valor de *deadline* absoluto de sua requisição que no escalonador servirá de base para a definição (segundo o *EDF [52]*) das prioridades de execução das requisições no servidor. Uma extensão do POA (*RT POA*: “*Real-Time Portable Object Adapter*”) no servidor é usada na gestão do enfileiramento das requisições segundo estas prioridades e na ativação de *threads* das *lanes* associadas a estas prioridades.

As especificações RT CORBA 1.1 [11] descrevem *threads* como entidades escalonáveis do sistema. Interfaces são especificadas para gerir *threads*, o que propicia uma grande portabilidade às aplicações. *Threads* podem ser definidas e controladas de

forma uniforme, independentemente do suporte de *threads* do sistema operacional e de seus mecanismos de escalonamento. As políticas de escalonamento são implementadas através da atribuição de prioridades das *thread* (prioridade das *lanes* a que estão associadas).

No protótipo implementado, de maneira semelhante aos trabalhos em [29] e [30], o modelo de invocação em tempo real assume que cada invocação iniciada de um método remoto é executada completamente mesmo depois do respectivo deadline absoluto. Neste caso, portanto, não há aborto na execução de requisições já iniciadas. Ou seja, como nos autores citados, fugimos das dificuldades de retrocessos no processamento distribuído, muito embora teoricamente no modelo não possamos assumir nenhum benefício na execução a *posteriori* do *deadline*. Mas diferente dos trabalhos citados assumimos o descarte de requisições com folga menor ou igual a zero. No modelo para a adaptação de parâmetros de QoS, são assumidas sinalizações de perda de deadline e de descarte de requisição. Estas sinalizações usam o suporte de exceções do CORBA e o próprio *reply* para que o interceptador do lado cliente possa acionar o objeto de QoS correspondente.

O escalonamento é efetuado pelos serviços de escalonamento em todos os nós servidores de forma transparente, disponibilizado assim as políticas de escalonamento atuantes sobre as invocações de métodos nestes servidores. *Pools* de *threads* e filas de invocações são pré-alocados no servidor para enfileirar e executar as requisições dos clientes. A seguir os componentes do modelo proposto terão seu funcionamento detalhado.

4.3.1 Mecanismos de Especificação e de Obtenção de QoS

Os requisitos de QoS são determinantes em tempo de *binding* quando os objetos e estruturas necessárias são instanciadas, especializando o suporte segundo os requisitos de QoS da aplicação. Os serviços de objeto CORBA serão ativados através de interceptadores em cada invocação de método, implementando as políticas relacionadas com os requisitos de QoS da aplicação.

As técnicas de especificação e de implementação de QoS, adotadas para descrever os requisitos de aplicação definindo parâmetros comunicação fim-a-fim, devem ser

flexíveis de modo a permitir que aplicações expressem seus requisitos de QoS na forma que mais lhe convier. Qualquer arquitetura que fornece esta flexibilidade apresenta mecanismos tanto estáticos quanto dinâmicos para a especificação de requisitos de QoS. É interessante também que esta flexibilidade incida sobre a representação dos parâmetros. Diferentes formatos de parâmetros de QoS devem ser suportados para que diferentes tipos de aplicação possam usar estes mesmos mecanismos.

A plataforma SALE fornece mecanismos tanto estáticos quanto dinâmicos para especificação de QoS. Os mecanismos estáticos de especificação de QoS tem como base a linguagem *QSL* (*Quality Specification/Scripting Language*), que se encontra em desenvolvimento. Ou seja, os requisitos de QoS são especificados de uma maneira semelhante a uma especificação IDL do CORBA. Estes requisitos são levados em conta na geração das estruturas necessárias para implementar a qualidade de serviço desejada na ligação cliente/servidor da aplicação. *Scripts* de mapeamento de parâmetros especificam como os diferentes formatos de parâmetros suportados são mapeados nos recursos do suporte.

Como desejamos que os requisitos possam ser modificados em tempo de execução, os objetos de QoS foram criados de modo a atuarem sobre os valores dos parâmetros de QoS e modificá-los dinamicamente segundo as necessidades da aplicação. Os objetos de QoS devem ser gerados a partir da tradução de uma especificação *QSL* associada a uma interface ou mais interfaces IDL. A necessidade de modificações dinâmicas se deve ao fato que parâmetros QoS da aplicação são dependentes da disponibilidade de recursos de sistemas operacionais e de suportes para comunicação em rede, em tempo de execução. Um exemplo dessas alterações dinâmicas é a adaptação executada pelo objeto de QoS baseado no histórico de invocações de um método ou no desempenho destas.

O objeto de QoS é uma interface onde cliente e servidor podem ter acesso ou modificar seus parâmetros de QoS através dos métodos mostrados na IDL da figura 4.3. Em seu objeto de QoS, o cliente pode atribuir, em tempo de execução, através do método *setParam*() da interface descrita, novos valores de parâmetros de QoS para suas chamadas remotas (por exemplo, o *timeout* de um método). No lado servidor, o objeto de QoS pode servir para estabelecer parâmetros locais como tempos de computação, atrasos,

características para a criação do *pool* de *threads* a ser utilizado pelo serviço de escalonamento, etc. Nós nos limitamos, neste trabalho, aos parâmetros temporais providos a partir dos requisitos do usuário-cliente.

```
struct Property{       // Do Serviço de Propriedades do CORBA
  string property_name;
  any    property_value;
};
typedef sequence<Property> Properties;
module SALE {           // IDL do SALE
  interface Qos {
      exception ParamEx { Properties qos_params; };
      attribute Properties myQos;
      void setParam (in string name, in any val)  raises (ParamEx);
      void getParam (in string name, out any val) raises (ParamEx);
      void configQos() raises (paramEx);
  }
}
```

Figura 4.3: IDL do Objeto QoS

Objetos QoS são localizados no nível de aplicação, juntamente com o objeto cliente e com o objeto servidor, permitindo que o suporte computacional seja configurado em tempo de ligação (*binding*) de modo a atender os requisitos da aplicação.

Os mecanismos de QoS definidos no modelo permitem que requisitos sejam associados a chamadas individuais, a todas as chamadas do mesmo método ou mesmo a todas as chamadas de um objeto. Os requisitos são definidos pelo servidor em relação à execução de seus serviços e pelo cliente sobre a chamada remota fim-a-fim, levando em conta os requisitos do servidor.

Esta abordagem apresentada é compatível com os modelos de objetos do RT-CORBA, FT-CORBA e do CORBA*Sec*, nos quais os objetos de serviço e os mecanismos de ORB que agem em tempo de execução nas chamadas de métodos remotos, são criados durante a operação de *binding* estabelecendo a conexão entre os objetos cliente e servidor. Os objetos de QoS são consultados via *interceptors*, durante a invocação de um método, no sentido de obter os valores dos requisitos e parâmetros de QoS, determinando que os mesmos sejam implementados enquanto a aplicação distribuída se executa.

4.3.2 Determinação de Parâmetros de Tempo Real: o Algoritmo de Definição

No modelo definido de QoS, as restrições de tempo real que devem ser verificadas nas chamadas de métodos remotos são determinadas a partir de requisitos de QoS definidos pelo cliente. No caso trabalhamos com o *timeout* definido pelo cliente nos seus requisitos de *QoS,* que determina a sua expectativa de duração da chamada de um método.

Até o momento foi apresentada a funcionalidade específica de cada componente do modelo proposto. Este item tem por objetivo apresentar o algoritmo (**figura 4.4**) que define as restrições de tempo real a partir de requisitos de QoS, definindo o comportamento destes componentes diante da necessidade de cumprir a expectativa do usuário, que é executar o método dentro do valor de *timeout*. A adaptação conduzida é feita no sentido de minimizar as perdas de deadline a partir do objeto cliente. O objeto de QoS do cliente centraliza os cálculos do algoritmo.

Inicialmente o usuário define o *timeout* para as execuções de um método remoto, através de uma especificação de, por exemplo, *QSL*. No cliente, a invocação de método remoto ativa por interceptação o objeto de QoS que calcula através do algoritmo da figura 4.4 os parâmetros de tempo real (no caso o *deadline* relativo) que definirão o atendimento do requisito de QoS do usuário.

A heurística proposta (**figura 4.4**), inicialmente define algumas variáveis de controle; é o caso do F e A que são pesos que determinam fatores de redução e adaptação nas equações do algoritmo. O *RTT* (Round Trip Time) define, tomando como base o relógio local ao cliente, o tempo para se completar a atual chamada remota (*round-trip* envolvido pela sincronização das mensagens de *request*/*reply*). O *Rmax* guarda o maior valor de *round-trip* nas sucessivas chamadas do método.

A rotina de iniciação define a atribuição inicial para o fator F do valor de 0,3. O coeficiente A – que define a adaptação deste fator F na ocorrência de exceções – tem atribuído o valor 0,9 na iniciação. Os valores iniciais de *RTT* e *Rmax* são assumidos como nulos.

Definição de Variáveis

T : *timeout*
D : *Deadline* relativo;
RTT : (*round-trip time*)
R_{max} : tempo máximo de resposta (*max_round-trip*), iniciado com *0*
F : fator de redução do *deadline* relativo; D/R – 0 < F < 1, iniciad0 com 0.3
A : coeficiente de ajuste do fator F, 0 < A < 1, iniciado em 0.9

Iniciação:

```
T ← QoS value;
F ← 0,3;
A ← 0,9;
Rmax ← 0;
RTT ← 0;
```

Envia mensagem:

```
{
  if (Rmax == 0)
    D = F * T / 2;
  else
    D = F * Rmax;
  set (T, request_id);
  set (D, request);
  envia (request, prioridade CORBA);
};
```

Recebe mensagem:

```
{
  if (resposta = sucesso)
    {calcula RTT;
     if (RTT > Rmax)
       Rmax ← RTT;};
  else
    {if (resposta = exceção)
      {if (exceção = timeout)
        {calcula RTT;
         Rmax ← RTT;
         sinaliza aplicação;
         exit ();};
      };
     else
      {if (exceção = deadline)
        F = F * A;
       else
        {if (exceção = folga_zero)
          {sinaliza aplicação;
           exit ();};};
      };
    };
};
```

Figura 4.4: Heurística usada na determinação de parâmetros de tempo real

No envio de uma requisição, se *Rmax* for igual a 0 (o que significa ser a primeira chamada deste cliente a um método específico), o *Deadline* relativo é assumido como igual à metade do *timeout* especificado pelo cliente multiplicado pelo fator de redução F, cujo valor inicial é 0.3. Ou seja, na primeira execução do método, com a expectativa do usuário de uma duração máxima da chamada de *timeout*, o *deadline* relativo será assumido como 15% do valor do *timeout*. O resto do tempo pode ser gasto na comunicação com as mensagens de *request*/*reply*. Se *Rmax* for diferente de zero, o valor do deadline relativo é definido por:

$$D_{relativo} = F * Rmax .$$

O *deadline* relativo, neste último caso, é assumido como 30% do valor do *Rmax*. Os outros 70% podem ser consumidos com as mensagens de *request*/*reply* referentes a atual chamada. Com o *deadline* relativo e o *timeout* definidos, o interceptador implementa os controles destes parâmetros no suporte de chamada remota. O *timeout* é controlado pelo objeto de QoS do cliente usando o tempo de seu relógio local. O *deadline* relativo descrito pela prioridade CORBA da mensagem de requisição enviada ao servidor.

A requisição chegando no servidor, o interceptador desvia a mesma para o cálculo do *deadline* absoluto do método adicionando o *deadline* relativo ao valor atual do relógio local. Esta é a "transformação" sofrida pela prioridade CORBA da requisição. O cálculo do *deadline* absoluto no servidor é uma necessidade no nosso modelo, uma vez que não está entre nossas premissas a sincronização de relógios ou o uso de servidores de tempo GPS[3]. A requisição é refletida no servidor para o serviço de escalonamento de tempo real para que este possa ordenar nas filas de requisição a invocação do método desejado, segundo a política EDF [52].

O *RT POA* é usado então para a ativação de *threads* das *lanes* de requisições. A prioridade de uma *thread* é definida pela *lane* a que está associada. Uma requisição ao ser ordenada em uma determinada *lane* vai ser executada no suporte de execução na prioridade da *lane*. O *pool* de *threads* usado é definido a partir do objeto de QoS do servidor onde estão especificados, além do *pool* de *threads*, o número de *lanes* estáticas ou/e dinâmicas, entre outras características. O processador é então ocupado, segundo uma escala construída por um escalonador *priority-driven*, sempre pela *thread* com maior prioridade.

Três tipos de exceções são previstos no nosso modelo para a execução remota: sinalização de perdas de *deadline* e sinalização de descarte de execução, ambas emitidas a partir do servidor, e não cumprimento de *timeout* emitida a partir do próprio suporte do cliente da chamada. Estes três tipos de exceção são tratados na rotina de recepção.

[3] Quando usados valores de *deadlines* absolutos determinados a partir do cliente, é necessário uma base de tempo global que permita uma visão coerente em todo o sistema sobre os valores de tempo. Um *Serviço de Tempo Global* pode ser feito disponível, a partir de interfaces seguindo o padrão CORBA [53]. Este serviço de tempo pode ser implementado em um sistema de larga escala (e disperso), usando receptores GPS e, por exemplo, o protocolo de sincronização de relógios *NTP* [56].

Na recepção do *reply*, a rotina de recepção (**figura 4.4**) do objeto de QoS testa se foi sucesso a execução do método remoto. Ou seja, se não ocorreu exceções. Se for o caso, as seguintes atualizações são feitas no objeto de QoS referentes ao *RTT* e o *Rmax*: o *round-trip* da chamada atual é calculado e se este supera o *Rmax*, o valor do *RTT* (o *round-trip* da chamada atual) é assumido como o novo valor do *Rmax*. A chamada é concluída retornando a aplicação cliente. No caso da resposta ter sido uma exceção, o algoritmo da figura 4.5 determina vários testes:

1. se a exceção for por exceder o valor de *Timeout* então, neste caso, o valor de *RTT* calculado é atribuído ao *Rmax* para fins de atualização, e uma exceção é enviada à aplicação cliente para que este possa redefinir, por exemplo, o *Timeout* de suas chamadas em um novo contrato;

2. se a exceção for por estouro do *Deadline* na execução do método remoto então, neste caso, o fator de redução F deve ser atualizado com: $F = F * A$, reduzindo o *Deadline* relativo da próxima chamada do método remoto;

3. se a exceção for por "folga zero" então, como conseqüência, a chamada não foi executada porque o *Deadline* relativo é menor ou igual ao tempo de computação do método correspondente. Neste caso não ocorre atualização das variáveis de controle e uma exceção é enviada a aplicação cliente.

A redução do *Deadline* relativo proposta em (2) implica, segundo a política *EDF [52]* implementada no objeto Serviço de Escalonamento do servidor, no aumento da prioridade das requisições subseqüentes do método associado. A diminuição de deadline pode ocorrer até que seja sinalizado pelo objeto servidor o descarte por folga menor ou igual a zero ($F \leq 0$). Neste caso e no de estouro de *timeout*, a aplicação cliente recebe sinalizações de *"tarefa não escalonável"* e *"timeout inadequado"*, respectivamente. O usuário pode refazer seus requisitos de QoS, propondo um *timeout* maior, recomeçando o processamento da heurística.

4.4 Mapeamento do Modelo no RT-CORBA

Foi apresentado até aqui o modelo de QoS proposto, sua heurística de definição de parâmetros e suas especificações. Neste item é apresentado um mapeamento mais fino dos componentes deste modelo no RT-CORBA.

4.4.1 Declaração da abordagem usada de prioridade CORBA

Como visto anteriormente neste capítulo (item 4.3), a política de prioridades adotada no modelo é do tipo *client-propagated*, de modo que o cliente deve defini-la antes de ativar o envio da mensagem de requisição ao servidor. A maneira como o cliente registra o tipo de prioridade CORBA é mostrada na **figura 4.5**. O estilo escolhido favorece a operação no servidor sobre os valores de prioridades CORBA.

```
CORBA ::PolicyList policies(1);
Policies.length(1);

Policies[0]= rtorb ->creat_priority_model_policy
(RTCORBA:: CLIENT_PROPAGATED, DEFAULT_PRIORITY);
// define a prioridade como CLIENT_PROPAGATED para o objeto que vai ser ativado

PortableServer::POA_var obj_var = root_poa->creat_poa;
(“obj_poa”, PortableServer::POAManager::nil(), policies);
// cria o POA com a política correta

obj_poa ->activate_object(my_obj);
// ativa o objeto “my_obj” no obj_poa Que foi criado com a política CLIENT_PROPAGATED
```

Figura 4.5: Declaração da política *CLIENT_PROPAGATED*

4.4.2 Mapeamento do interceptador do cliente

Inicialmente, numa requisição, o interceptador do lado cliente é acionado, refletindo o fluxo da chamada para o objeto de QoS do cliente, onde o valor do *timeout* está armazenado e o cálculo do *deadline* relativo da chamada é realizado. Uma vez estabelecido estes parâmetros de tempo real a requisição é enviada. A **figura 4.6** mostra parte da rotina de envio implementada no objeto de QoS chamada a partir do interceptador pelo ORB.

```
void send_request (ClientRequestInfo request_info) {

  Start = System.CurrentTimeMillis();      // Pega o tempo inicial do envio da
                                           // tarefa (para controlar o timeout)
  ObjQoS->updateDeadline();                // Calcula o deadline para esta chamada
}
```

Figura 4.6: Interceptador no cliente, antes do envio da chamada

Já quando o interceptador é acionado pelo ORB na recepção de uma resposta do servidor, outro método do objeto de QoS é ativado. Como duas possibilidades são distinguidas no algoritmo da **figura 4.4**: a com sucesso e a com exceção. Estes dois comportamentos distintos são implementados separadamente por dois métodos do objeto de QoS. O primeiro (**figura 4.7**) é implementado para o caso da resposta ser de sucesso da chamada, enquanto o segundo (**figura 4.8**) é implementado para o caso da resposta ser uma exceção.

```
void receive_reply (ClientRequestInfo ri) {
                                           // Armazena na variável RTT o tempo do
  RTT = System.CurrentTimeMillis() – Start;  // round trip da mensagem medido pelo
                                           // tempo de partida subtraido do tempo
                                           //de chegada.
  ObjQoS->updateRmax(RTT);                 // Atualiza o maior round trip (Rmax)
                                           // comparando o atual com o RTT medido.
}
```

Figura 4.7: Interceptador do cliente, caso de sucesso da chamada

```
void receive_exception(ClientRequestInfo ri) raises (ForwardRequest)  {
   unsigned long RTT = currentTimeMillis() – start;   // Armazena o round trip
   ObjQoS->handleException(ri.exceptions(), RTT); // Trata a exceção
}
```

Figura 4.8: Interceptador do cliente, caso de exceções.

Os métodos do objeto de QoS updateDeadline(), updateRTT() e handleException(), cujo código é listado na **Figura 4.9**, são usados pelos interceptadores do cliente para implementar a heurística especificada na **Figura 4.4**. É importante ressaltar

que estes métodos não estão disponíveis na interface do objeto de QoS com a aplicação, mostrada na **Figura 4.3**, e desta forma podem ser acessados apenas pelos interceptadores.

```
void updateDeadline () {
   if  (Rmax == 0)          // indica primeira chamada a um servidor específico
    D = F * T/2;            // calculo baseado no timeout
  else
    D = F * Rmax;           // calculo baseado no maior round-trip time (Rmax)
}

void updateRmax (unsigned long RTT) {
  if (Rmax < RTT)           // se o RTT atual foi maior que o máximo registrado anteriormente
    Rmax = RTT;             // fazer dele o novo RTT máximo
}

void handleException(TypeCode[] ex, unsigned long RTT) {
   for (int k=0; k<ex.length(); k++) {
  {
     if (ex[k].name().equals("DeadlineException")           // diminui o fator F em caso de
          F = F * A;                                        //  estouro de deadline
     else if (ex[k].name().equals("TimeoutException")       // atualiza o Rmax com o RTT no qual
          updateRmax (RTT);                                 // o timeout estourou, avisa o usuário
          return ();                                        // para redefinir timeout
     else if (ex[k].name().equals("SchedulingException"); // deadline menor que o tempo
          return ();                                        // de computação da tarefa
   }
}
```

Figura 4.9: Métodos do Objeto de QoS usados pelo interceptador do cliente.

4.4.3 Escalonador

O escalonador, examinando o deadline absoluto da requisição que chega, deve definir uma nova distribuição das requisições existentes nas *lanes* junto com a recém chegada. A **figura 4.10** mostra o módulo *EDF_SCHEDULING*[40] construído para a distribuição das requisições nas *lanes* definidas e, por conseqüência, alocando às *threads* associadas para execução. As operações disponíveis no escalonador para tais operações são vistas na figura 4.10.

```
Module EDF_Scheduling{
        struct SchedulingParameter      {           // cria uma estrutura que lê o deadline e
                        time base::timet Deadline;      // uma possível importância
                        long importance;
        };
        local interface SchedulingParameterPolicy : CORBA::policy{
                        Attribute SchedulingParameter value;      // atribui um valor(prioridade) a
                                                                  // uma estrutura
        };
        local Interface Scheduler:RTScheduling::Scheduler{
                        SchedulingParameterPolicy Create_Scheduling_parameter
                (in SchedulingParameter value);  // utiliza o SchedulingParameterPolicy para criar
        };                                       // um parâmetro a ser usado pelo escalonador
};
```

Figura 4.10: Módulo *EDF_SCHEDULING*[40]

4.5. Conclusão

Neste capítulo, foi descrita a proposta de nosso modelo computacional que suporta as funcionalidades de QoS definidas dentro do projeto SALE. No modelo apresentado nos fixamos em requisitos referentes a desempenho (*timeout* de chamadas). Mas as considerações feitas em relação a atender as necessidades de tempo real podem ser estendidas para outros requisitos como de segurança e de tolerância a falhas.

O modelo computacional e ferramentas adicionais para o suporte de QoS foram apresentados neste texto e discutidos no seu mapeamento no RT-CORBA. O texto também explora a definição de parâmetros em função de um requisito de *timeout* de chamada. A atuação dinâmica através de objetos de QoS é apresentada, mostrando o algoritmo que define esta atuação sobre os parâmetros na tentativa de manter a qualidade de serviço desejada.

Os resultados experimentais obtidos a partir desta proposta são objeto de discussão do capítulo que se segue.

5. Avaliação da Proposta

No capítulo 4 foi apresentada a proposta de um modelo de suporte de QoS para aplicações de tempo real. O modelo visa adaptar o *deadline* perdido de uma requisição para que, em uma próxima chamada, esta possa ser escalonada.

Procurando validar o modelo, este capítulo tem por objetivo apresentar as simulações feitas com o seu uso, fazer a análise dos resultados obtidos e apresentar uma situação real de uso do modelo.

5.1 Simulações da heurística

Para avaliar o funcionamento do modelo foram realizadas diversas simulações, sem no entanto considerar um cenário específico de aplicação de tempo real para ser utilizado como objeto de teste. Nas simulações realizadas, foi empregado um simulador escrito em C++ desenvolvido no Laboratório de Controle e Micro-Informática (LCMI) do Departamento de Automação e Sistemas (DAS) da Universidade Federal de Santa Catarina (UFSC), pelos professores Carlos Montez e Rômulo de Oliveira, rodando em um Pentium Celeron 500 Mhz com 64 MB de memória RAM, e adaptado às características da heurística contida no modelo proposto. Neste ambiente, foi criado um número de tarefas periódicas que tentavam ser executadas em um servidor respeitando seus deadlines.

Estas tarefas tinham definidos os seguintes atributos:

- *Período* – o qual define a regularidade com que as tarefas eram enviadas;
- *Timeout* – define o tempo de espera de resposta pelo usuário;
- *Deadline* – é o *deadline* relativo, o qual define o tempo que a tarefa tem para ser executada;

- *Jitter* – define o tempo gasto no envio de uma tarefa do cliente até o servidor; quando o jitter é 0 significa que a invocação da tarefa é local;

- *Tempo de computação* – é o tempo que uma tarefa precisa ocupar o processador para executar sua rotina.

Foram definidas também uma função cujo objetivo é adaptar o deadline quando este se esgota, assim como uma função para calcular e atualizar o maior *round trip time* (ou seja, o maior tempo de espera do cliente por uma resposta do servidor da tarefa). Estas funções são necessárias para o perfeito funcionamento da heurística.

Para simular o *pool* de threads foi implementada uma fila que é atualizada a cada chegada de tarefas e que em seu topo sempre tinha a tarefa cujo deadline estava mais próximo de se esgotar. Desta forma, fazendo com que a tarefa a ser executada fosse sempre a do topo da fila, foi garantida a política de escalonamento *EDF*.

Na simulação, quando existe a ocorrência de uma das exceções previstas no modelo, esta é mostrada na tela para que o usuário tenha conhecimento de seu acontecimento, e no caso de estouro de timeout ou folga_zero, a simulação é interrompida.

No item a seguir são mostradas as variações dos modos de simulações executados, assim como uma análise dos seus resultados.

5.2 Análise dos resultados obtidos

Após ter definido o ambiente de simulação e adaptado-o ao modelo proposto, simulações foram feitas usando um conjunto de tarefas que, em uma situação de pior caso, levaria a perdas de deadline. Uma tarefa deste conjunto foi definida como remota, ou seja, com *jitter* diferente de 0. Esta tarefa foi utilizada para o controle individual de perdas de deadline visto que ela, por chegar sempre após as demais tarefas e portanto possuir um deadline relativo maior, teria seu primeiro deadline estourado nas simulações sempre que houvesse uma carga excessiva no processador. No restante dos dados, as simulações foram feitas de diferentes formas, tendo como entrada diferentes atrasos de envio e também

diferentes valores de computação. A seguir serão discutidas individualmente estas simulações.

5.2.1. Simulação com atraso de envio e tempo de computação fixos

Nesta simulação, os atributos foram considerados constantes, com a finalidade de verificar o comportamento da heurística sob a carga máxima para aquele conjunto de tarefas. Desta forma, o tempo de computação das tarefas foi definido em 50 unidades de tempo (u.t.) e o momento de sua chegada (*jitter*) em 100 u.t. para a tarefa remota e em 0 nas demais.

Com a definição destes valores feita previamente, a carga pôde ser caracterizada como **estática** e **limitada**, e pôde ser garantida uma *previsibilidade determinista* com relação ao conjunto das tarefas simuladas, que seguiram a abordagem *best-effort*.

Como resultado tivemos uma igualdade nas perdas de deadline em comparação com o escalonamento EDF (**gráfico 5.1**). No entanto, a quantidade de perdas de deadline individual da tarefa proposta com *jitter* diferente de 0 foi consideravelmente maior ao utilizar somente o algoritmo “EDF”, quando houve perda de 100%, do que ao utilizar o modelo proposto com adaptação de seus parâmetros, quando ocorre perda de deadline somente na primeira chamada da tarefa(**gráfico 5.2**).

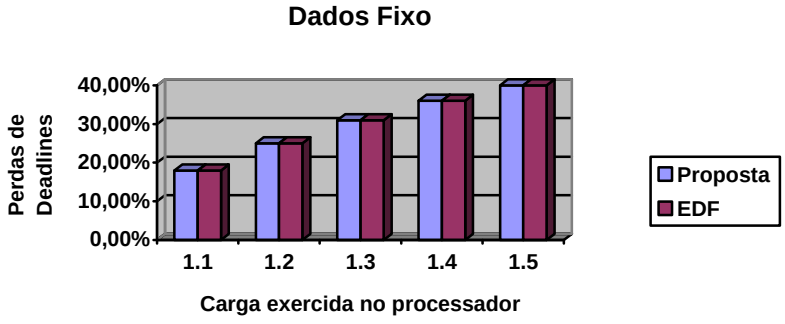

Gráfico 5.1 – Perdas de deadline com atraso e computação fixos.

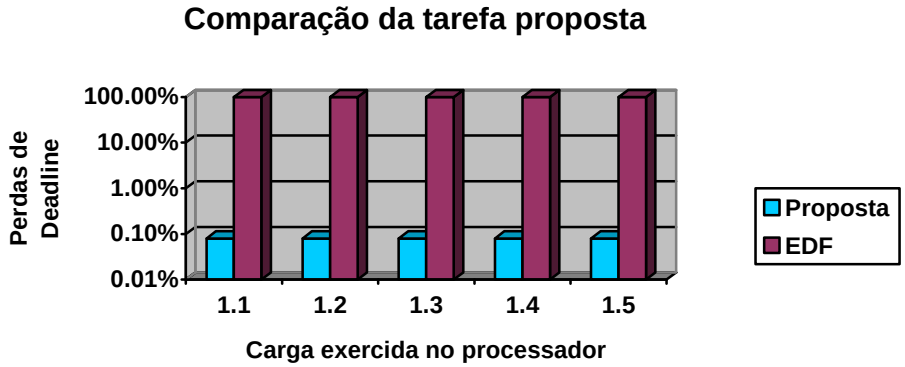

Gráfico 5.2 – Perdas de deadline da tarefa proposta com jitter, na situação analisada.

5.2.2 Simulação com atraso de envio variável e com tempo de computação fixo

Nesta simulação, o tempo de atraso no envio da mensagem (*jitter*) foi submetido a uma função de distribuição exponencial para que tivesse seus valores tendendo a 100 u.t., mas variando para que as diferentes cargas em uma rede de longa distância conseguissem ser simuladas. Já o tempo de computação das tarefas foi fixado em 50 u.t. para evitar que em momentos de baixos valores de *jitter*, o processador não ficasse ocioso durante um tempo na parte final de um período.

Devido à definição dos valores de envio da tarefa com *jitter* diferente de 0 não poder ser feita previamente, a carga foi caracterizada como **dinâmica** e **ilimitada**, e pôde ser garantida uma *previsibilidade probabilística* do conjunto das tarefas simuladas, que também seguiram a abordagem *best-effort*. Os resultados obtidos são apresentados (**gráficos 5.3 e 5.4**) e analisados a seguir.

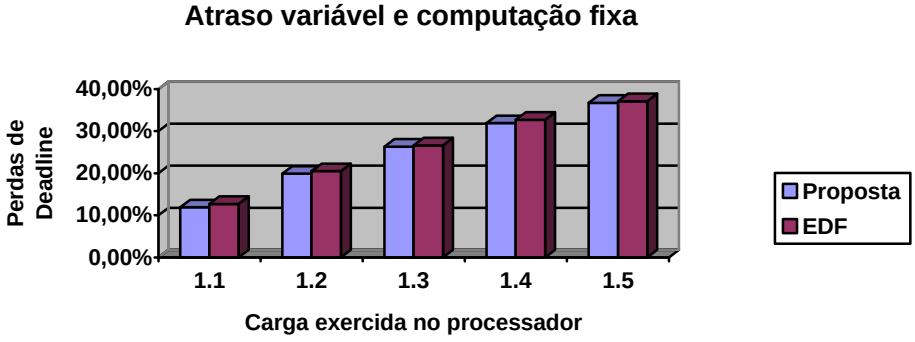

Gráfico 5.3 – Perdas de deadline com atraso variável e computação fixa.

O **gráfico 5.3** mostra a quantidade de perdas de deadline em %, em diferentes cargas de solicitação do processador, com a aplicação do modelo de escalonamento “EDF” puro e com a aplicação do modelo proposto. Na análise feita, é possível afirmar que quanto a melhorias nas perdas de deadlines no conjunto completo de tarefas, o modelo proposto apresenta mínima melhora quanto ao “EDF” puro (**gráfico 5.3**), mas se observado o resultado das perdas individuais quanto a tarefa de teste (**gráfico 5.2**), esta apresentou uma perda de 100% quando é usado escalonamento “EDF”, enquanto que quando aplicado o modelo proposto ela só apresentou perdas nos primeiros envios. Esta diminuição nas perdas individuais com o uso do modelo proposto deve-se ao uso da heurística de diminuição de deadline para aumento de prioridade, que fez com que a tarefa se tornasse mais prioritária e consequentemente passasse a ter acesso ao processador.

5.2.3. Simulação com atraso de envio e tempo de computação variáveis

Nesta simulação, o tempo de computação das tarefas foi definido como variável entre 20 u.t. e 50 u.t. (pior caso) e o tempo de atraso no envio da mensagem (jitter) foi novamente submetido a uma função exponencial para que tivesse seu valor variando, mas tendendo a 100 u.t.

Com a definição dos valores dos dados sendo variável, a carga das tarefas foi caracterizada, como na simulação anterior, como sendo **dinâmica** e **ilimitada**, e uma previsibilidade probabilística das tarefas simuladas pôde ser garantida.

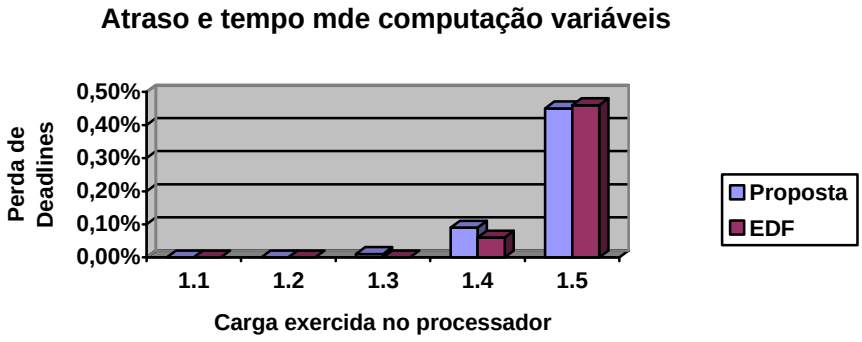

Gráfico 5.4 – Perdas de deadline com atraso e computação variáveis

Como resultado desta simulação tivemos um baixo nível de perdas de deadline (**gráfico 5.4**) devido ao fato do conjunto de computação das tarefas simulado não ter atingido 100% de ocupação no processador, mas novamente a tarefa remota teve suas perdas minimizadas quando o modelo proposto se encontrava em uso (**gráfico 5.2**).

A carga real do processador é apresentada na **tabela 5.1** para as respectivas cargas de pior caso geradas.

Carga de pior caso (em %)	Ocupação real do processador (em %)
110 %	70 %
120 %	76 %
130 %	83 %
140 %	90 %
150 %	95 %

Tabela 5.1 – Comparação das cargas real e de pior caso (em %)

5.3. Considerações sobre os resultados de simulação e o modelo proposto

Objetivo neste trabalho foi estabelecer um suporte de QoS que permitisse ao usuário descrever seus requisitos. O suporte proposto determinaria então os parâmetros de tempo real no sentido de atender estes requisitos. Ao contrario das propostas como a do APMC [29] e a do um controlador proporcional variando os períodos das tarefas [30] (que atuam basicamente no tratamento de sobrecargas em servidores na execução das invocações remotas de métodos), o nosso modelo não foi concebido para tratar com o controle de sobrecarga do servidor. As propostas citadas trabalham com a idéia de diminuir a qualidade de serviço para tratar os problemas de sobrecarga no servidor.

Nas simulações que executamos, tanto no caso da heurística apresentada como no EDF, foram feitos testes de folga positiva antes da ativação das tarefas (*threads*). Numa execução de *best-effort* com o EDF, sem o teste de folga, alguns autores consideram o mesmo como uma política instável para o trato de carga dinâmica [55] .

As conclusões que chegamos sobre os testes realizados neste trabalho é que os resultados de simulação mostram a melhora da escalonabilidade da tarefa com o deadline ajustado (o que pode parecer evidente), em todos os casos considerados. A adaptação do deadline para atender o requisito de QoS se mostra efetivo. Outra conclusão é que as modificações sobre os parâmetros temporais desta tarefa não se refletem em uma piora no desempenho da política EDF sobre o conjunto de tarefas considerado.

As escolhas dos fatores e parâmetros usados na simulação não foram ao acaso. Mas determinados a partir de um conjunto de testes. O *Rmax*, por exemplo, nos testes apresentados neste texto assume sempre o pior caso de *round-trips*. Outras possibilidades usadas foram: o uso de uma janela de chamadas consecutivas onde o valor de *Rmax* foi assumido como o valor máximo na janela; e, na mesma janela, o uso de uma média ponderada dos valores de *round-trip* (privilegiando as últimas na média, como é feito no protocolo TCP) como valor de *R* (que deixa de ser o máximo). Nós nos fixamos em *Rmax* sem limites de uma janela porque capta a incerteza do suporte de comunicação e permite que os valores de deadline que são inferidos como 30% de *Rmax* ficasse subdimensionado com alguma freqüência devido a esta não previsibilidade nas comunicações. Ou seja, escolhemos a linha de pior situação conhecida para o *Rmax* no sentido de minimizar erros na escolha de *deadline*.

O objetivo na nossa proposta é atender o contrato com o usuário-cliente. A adaptação de parâmetros define a variação do *deadline* relativo de um método entre valores limitados em um terço do *timeout* e o tempo de computação de pior caso do método em questão (valor estimado). Enquanto o método não é escalonado, seu *deadline* vai sendo ajustando no intervalo assumido. Nenhuma simulação foi feita comparando a nossa heurística com outras, principalmente as citadas neste texto, porque essas estão baseadas em princípio diferente: são dirigidas essas últimas ao problema de sobrecarga do servidor.

Uma das dificuldades que facilmente poderia ser apontada é que, se todas as tarefas estão executando segundo esta heurística, pode haver um aumento geral de prioridades e que isto poderia não garantir uma distribuição *justa* do processador e muito menos ter uma

certa probabilidade de atender o cliente. Não foi nossa intenção a procura de uma distribuição *justa* do processador entre as tarefas do servidor.

Uma maneira de evitar o aumento generalizado de prioridades de execuções no servidor é estabelecer um histórico de execuções, a partir do objeto de QoS do cliente, que definiria um número de execuções corretas em uma janela considerado como aceitável no sentido de manter a qualidade de serviço. A exemplo do (*m, k*)-*firm* [43] procuramos então estabelecer o que é aceitável em termos de perda na execução do método remoto em uma janela de invocações consecutivas. Logo, quando nesta janela for atingido o valor de execuções corretas desejáveis podemos manter o *deadline* relativo com o seu valor fixo mesmo diante de perdas de deadline em invocações subseqüentes. Podemos também em casos mais críticos de sobrecarga efetuar acréscimos no deadline do método remoto na janela, uma vez atingido o limite aceitável de execuções corretas.

Realizamos testes de simulação para o caso de uma tarefa atuando segundo o deadline ajustável e o histórico de uma janela. Nestes testes realizados definimos uma janela de 10 ativações do método remoto com 7 execuções com sucesso. Quando atingido o limite de execuções com sucesso na janela, nos insucessos subseqüentes, a heurística ou não diminuía o deadline (um dos testes) ou aumentava o deadline multiplicando pelo fator 1/F (outro teste realizado). Mas, os resultados obtidos neste testes não foram considerados significativos para a melhora da escalonabilidade do conjunto ou mesmo da própria tarefa onde se fazia o ajuste do *deadline*; por isto não apresentamos estes testes neste texto.

Se estendêssemos o nosso problema para melhorar a escalonabilidade do conjunto de tarefas no processador do servidor, isto representaria que teríamos que modelar todas as tarefas segundo a nossa heurística e fazer o controle de histórico. E, para verificarmos a efetividade da mesma, teríamos que conduzir um estudo comparativo (envolvendo simulações provavelmente) com abordagens como as apresentadas em [29] e [30]. O tempo que tínhamos para o termino da dissertação foi um limitante para que não estendêssemos os trabalhos também nesta direção. De qualquer forma, é bom que se enfatize que a relaxação da expectativa de execuções com sucesso em uma janela de chamadas consecutivas para melhorar a sobrecarga representa também numa negociação entre a perda de qualidade de serviço e o tratamento da sobrecarga no servidor.

O problema de sobrecarga poderia ainda ser tratado de uma maneira mais adequada usando o modelo de QoS. Por exemplo, poderíamos definir o uso da técnica de escalonamento adaptativo (*m*, *k*)-*firm* no servidor. O uso desta técnica implicaria na obtenção de certos parâmetros (com os valores de *m* e *k*, por exemplo) de requisitos descritos no objeto de QoS do servidor. A implementação desta técnica de escalonamento poderia se dar junto com a que foi definida no texto para atender o cliente neste trabalho. A idéia básica disto é que o cliente continuaria atuando nos *deadlines* relativos nos insucessos, enquanto o servidor aplicaria o histórico na evolução das prioridades de *threads*. Porém a abordagem de *deadlines* e a do histórico controlado no servidor do (*m*, *k*)-*firm* não são exatamente complementares porque partem de princípios diferentes em relação à qualidade dos serviços oferecidos. A diminuição do *deadline* relativo para o método remoto no cliente não implica na definição pelo (*m*, *k*)-*firm* de uma prioridade mais alta no lado do servidor. Um estudo mais profundo é necessário para compatibilizar estas técnicas de adaptação.

As propostas do APMC [29] e de um controlador variando os períodos das tarefas [30] parecem a nós menos adequadas para ambientes dinâmicos como a Internet que a apresentada neste texto, porque preconizam execuções periódicas de seus métodos. Estas, embora tratem as sobrecargas com variações de prioridades, são a princípio baseadas em políticas de prioridades fixas. As duas propostas citadas não possuem suporte de QoS.

O suporte de QoS proposto neste texto, a exemplo do TAO [13] e do *QuO* [25] não modifica ou estende as especificações CORBA. No entanto, TAO exige que os requisitos de QoS sejam especificados estaticamente em tempo de compilação no código da aplicação. QuO também requer especificação estática de QoS, usando a linguagem QDL. Já na nossa proposta é possível a adaptação dinâmica dos parâmetros de tempo real utilizados com o objetivo de influenciar a forma como o servidor executa as chamadas de métodos tendo em vista as variações no ambiente de execução e no meio de comunicação.

5.4 Exemplo de aplicação: Empresas Virtuais

Devido ao fato de que as simulações apresentadas até aqui terem sido feitas sem a definição de um cenário específico de caso de uso, este item apresenta um exemplo de uso do modelo proposto em uma empresa virtual (E.V.).

Empresa virtual é o conceito dado à união temporária de empresas de forma a colaborar na obtenção de um produto final. Neste tipo de organização, núcleos originalmente independentes se unem em busca de um objetivo comum, trocando informações de modo a coordenar as suas ações.

Uma empresa virtual envolve, em sua produção, uma gama de tecnologias as quais se utilizam de diversos recursos de hardware e software que, além de serem muitas vezes heterogêneos, estão distribuídos em sistemas computacionais em diversos locais. Tendo em vista esta distribuição, o conceito de empresas virtuais define que as empresas devem utilizar mecanismos de comunicação para que haja troca de informações e sincronia eletrônica em suas produções.

Ferramentas de escalonamento de atividades são necessárias para que as ações das empresas envolvidas no processo de produção sejam efetuadas de forma coordenada, enquanto ferramentas de colaboração devem ser usadas para executar trabalho cooperativo. Outros serviços de mais alto nível, tais como os de suporte ao projeto colaborativo de produtos e a monitoração em tempo real da entrega de encomendas, requerem infra-estruturas extremamente robustas, seguras e eficientes para que sejam utilizados pelas empresas. No nível administrativo, uma série de ferramentas deve estar disponível para que a cooperação entre empresas possa ocorrer de maneira efetiva.

O sistema HOLOS-MASSYVE é um exemplo de suporte computacional para operação de empresas virtuais [41]. O HOLOS-MASSYVE consiste em um sistema

multiagente[4] que atua como intermediário no escalonamento das necessidades da empresa virtual, nos equipamentos disponíveis do chão de fábrica de uma empresa componente da empresa virtual. Em momentos de indisponibilidade de um certo equipamento, o sistema HOLOS-MASSYVE foi desenvolvido para suportar uma negociação entre seus agentes a fim de criar, dinâmica e temporariamente, a melhor disposição dos equipamentos para um plano de produção específico. Devido aos requisitos temporais existentes nos sistemas de manufatura, o sistema HOLOS-MASSYVE necessita de uma comunicação baseada em requisitos de tempo real com os equipamentos. Esta comunicação possibilitará que o sistema corrija em tempo possíveis problemas, mantendo assim, coerente o plano de produção traçado.

As restrições temporais utilizadas pelos agentes HOLOS-MASSYVE em sua comunicação com os equipamentos são definidas pela plataforma na execução de suas chamadas utilizando o suporte RT-CORBA. Os requisitos de QoS utilizados nos agentes HOLOS-MASSYVE são expressos pelos seguintes parâmetros:

- *Timeout* – tempo máximo de espera do cliente pelo retorno da chamada;
- *Deadline* – tempo máximo de duração da execução do método no servidor;
- *Priority* – prioridade de execução do método no servidor.

O valor do *timeout* é obtido a partir da especificação do cliente. Por exemplo, o timeout das operações executadas por um controlador da célula de manufatura é especificado usando uma linguagem de especificação de requisitos de QoS, chamada QSL (*QoS Specification Language*)[49], conforme mostrado na **figura 5.1**.

```
#include “RT-CORBA.QSL”;          // Define parâmetros de QoS do RT-CORBA
QoS Controller: RT-CORBA {       // Objeto Controller usa o suporte RT-CORBA
start_program::timeout = 0.005;  // Timeout para iniciar a operação (5ms)
stop_program::timeout = 0.005;   // Timeout para parar a operação (5ms)
};
```

Figura 5.1 – interface em QSL de um agente HOLOS-MASSYVE

[4] Basicamente um sistema multiagente é uma agregação de processos computacionais (“agentes”) que, com base em variados graus de autonomia, interagem visando alcançar um objetivo comum [35].

As chamadas são efetuadas com o *timeout* definido pelo cliente que, seguindo o modelo de funcionalidade proposto no item 4.4, é utilizado para calcular o *deadline* relativo quando a chamada é interceptada ainda no lado cliente. Então este *deadline* será inserido no contexto da chamada quando esta for enviada ao servidor.

Já no lado servidor, como também é descrito no item 4.4, a requisição é interceptada e desviada para o serviço de escalonamento de tempo real, que utiliza a política de escalonamento *EDF*. O serviço de escalonamento gera uma prioridade para ocupar o campo destinado àprioridade CORBA, baseado no *deadline* especificado, do qual o POA se utilizará para definir a prioridade da *thread* que a requisição ocupará. Caso ocorra alguma exceção por estouro de *deadline*, *timeout* ou *folga_zero*, estas serão tratadas no interceptador do cliente conforme definido pela heurística proposta.

5.5 Considerações finais

Tendo em vista validar o modelo proposto no capítulo 4, este capítulo apresentou um conjunto de simulações onde tarefas com diferentes valores de computação e de tempo de envio tiveram suas execuções simuladas e como resultado foram coletadas e apresentadas as porcentagens de perda de deadline destas.

As tarefas foram simuladas com dois tipos de modelo de escalonamento, um usando o algoritmo “EDF” puro e outro baseado no modelo proposto no Capítulo 4. A análise foi feita com relação aos resultados de perda de deadline tanto do conjunto inteiro de tarefas quanto das perdas individuais da tarefa remota, a qual apresentava um atraso de envio.

Os resultados obtidos serviram para mostrar que com a utilização do modelo proposto, um cliente consegue diminuir suas perdas individuais de deadline, conseguindo assim uma maior confiabilidade de suas tarefas no que diz respeito aos seus requisitos temporais.

Ao final deste capítulo, foi descrito um exemplo prático de utilização da heurística proposta em um suporte para operação de empresas virtuais. Este exemplo mostra como requisitos temporais de uma aplicação prática podem ser utilizados no modelo proposto para escalonamento das atividades de um sistema de manufatura de uma empresa virtual.

No capítulo seguinte serão apresentadas algumas considerações finais sobre este trabalho, bem como algumas propostas de trabalhos futuros.

6. Conclusões e trabalhos futuros

Devido à necessidade de reestruturação das empresas, com o intuito de melhor adaptar-se a globalização apresentada pelo mercado nos últimos tempos, Empresas Virtuais surgem como um conceito de união temporária de empresas as quais devem interagirem entre si para a obtenção de um produto final. Neste tipo de organização, núcleos originalmente independentes se unem em busca de um objetivo comum, trocando informações de modo a coordenar as suas ações.

Sistemas de tempo real são aqueles que apresentam responsabilidades temporais quanto aos resultados obtidos, estas responsabilidades são expressas na forma de prazos a serem cumpridos que acarretam em mudanças em seu comportamento, fazendo com que o sistema não execute somente com o comprometimento lógico das tarefas mas também com responsabilidade quanto aos valores temporais de suas execuções.

QoS é um conjunto de características que eqüivalem a fatores como, compromisso de tempo de resposta, desempenho, confiabilidade, qualidade de saída, entre outros, necessários para atender os requisitos de aplicações. Dentre as aplicações com requisitos de QoS podemos destacar os sistemas multimídia, tempo real e de trabalho cooperativo.

O CORBA foi definido pela OMG como uma arquitetura para detalhar as interfaces e as características do componente a fim de oferecer uma solução fácil para o desenvolvimento de aplicações distribuídas, ou em outras palavras, um mecanismo de software que permita a comunicação dos objetos, independentemente de suas plataformas ou modo como foram implementados. Em complemento ao CORBA, extensões foram desenvolvidas para adicionar a sua arquitetura mecanismos de tempo real, segurança, tolerância a falhas, etc. O RT-CORBA é um grupo destas extensões do CORBA que foi especificado para adiciona mecanismos de Tempo real ao seu núcleo.

Neste trabalho, foram relatados alguns trabalhos que em suas propostas envolvem o uso do RT-CORBA e dos conceitos de QoS, os quais serviram de base para o desenvolvimento da proposta apresentada.

A proposta apresentada foi de um modelo computacional desenvolvido para possibilitar à arquitetura SALE atender as necessidades de tempo real das empresas virtuais. O modelo foi desenvolvido com a utilização das especificações do RT-CORBA para a definição de alguns componentes de sua estrutura, assim como a utilização de especificações de QoS para capturar as necessidades do usuário.

A validação do modelo foi apresentada por meio de simulações feitas em um simulador desenvolvido no Laboratório de Controle e Micro-informática (LCMI) do Departamento de Automação e Sistemas (DAS) da Universidade Federal de Santa Catarina (UFSC), pelo professor Carlos Montez para sua tese de doutorado, e adaptado as características da heurística contida no modelo proposto. Neste ambiente, um número de tarefas periódicas eram estabelecidas e tentavam executar em um servidor respeitando seus deadlines. Os resultados obtidos foram referentes a porcentagem de perdas de deadlines ocorridas, que, quando analisados em uma tarefa específica, foram menores com a utilização do modelo proposto do que com a política “EDF”.

Como trabalhos futuros, a relação abaixo é indicada:

- Implementação e testes da heurística proposta neste trabalho no ORB para tempo real “TAO”;

- Especificação do suporte em uma linguagem de especificação de QoS (QSL, por exemplo);

- Integração com os trabalhos desenvolvidos pelo grupo de pesquisa nas áreas de tolerância a falhas e segurança, de modo a obter um suporte computacional capaz de observar os requisitos de desempenho, confiabilidade e segurança de aplicações em sistemas distribuídos de larga escala;

- Desenvolvimento de aplicações voltadas a empresas virtuais com base neste Suporte.

7. Referências

[1] Steve Vinoski. “CORBA: Integrating Diverse Applications Within Distributed Heterogeneous Environments ”. Revista IEEE, vol 35, nº 2, Fevereiro de 1997.

[2] Douglas C. Schmidt. “Developing Distributed Object Computing Applications With CORBA ”. http://www.cs.wustl.edu/~schmidt/PDF/corba4.pdf

[3] Frank Siqueira. “ CORBA – Common Object Request Broker Architecture “. Em nota técnica do Laboratório de Controle e Micro-Informática da Universidade Federal de Santa Catarina. Florianópolis, SC, Brasil.

[4] Rafael R. Obelheiro. “*Modelos de Segurança Baseados em Papéis para Sistemas de Larga Escala: A Proposta RBAC-JaCoWeb* ”. Dissertação de mestrado, Programa de pós-graduação em Engenharia Elétrica, Universidade Federal de Santa Catarina, Florianópolis, SC, fevereiro de 2001.

[5] Disponível em:
http://www.omg.org/technology/documents/formal/naming_service.htm

[6] Disponível em:
http://www.omg.org/technology/documents/formal/transaction_service.htm

[7] Disponível em :
http://www.omg.org/technology/documents/formal/event_service.htm

[8] Disponível em :
http://www.omg.org/technology/documents/formal/c++.htm

[9] Disponível em :
http://www.omg.org/technology/documents/formal/corba_iiop.htm

[10] Joni Fraga, Lau Cheuk, Carla Merkle e Carlos Montez. "*Suporte para Aplicações Críticas nas Especificações CORBA: Tolerância a falhas, Segurança e Tempo Real",* 19° Simpósio Brasileiro de Redes de Computadores, 21 a 25 de maio de 2001, Florianópolis, Brasil.

[11] Especificações RT-CORBA versão 1.1, agosto de 2002. http://www.omg.org

[12] Douglas Schmidt e Fred Kuhns. "An *Overview of the Real-time CORBA Specification"*. Revista IEEE junho de 2000.

[13] Douglas Schmidt. " TAO: A Pattern-Oriented Object Broker for Distributed Real-time and Embedded Systems ". http://dsonline.computer.org/middleware/articles/dsonline-TAO.html

[14] Carlos Mitidieri. " TAO – Implementação e Modelo de Programação". http://www.delet.ufrgs.br/~miti/Disciplines/CMP167/t1_teoria_tao/

[15] Douglas Schmidt. "Object Interconnections: Real-time CORBA, Part 2: Applications and Priorities". http://www.cuj.com/experts/2001/vinoski.htm

[16] Douglas Schmidt. " Techniques for Enhancing Real-time CORBA Quality of Service ". http://www.cs.wustl.edu/~schmidt/PDF/IEEE-proc.pdf

[17] Disponível em:
http://www.cs.wustl.edu/~schmidt/

[18] Douglas Schmidt, David L. Levine e Sumedh Mungee. "The Design of the TAO Real-Time Object Request Broker". Em Computer Communications, Elsivier Science, Volume 21, Numero 4, Abril, 1998.

[19] OMG - Object Management Group. http://www.omg.org

[20] Frank Siqueira e Vinny Cahill. “Quartz: A QoS Architecture for Open Systems”. In Proceedings of the 20th IEEE International Conference on Distributed Computing Systems - ICDCS'2000, Taipei, Taiwan, April 2000.

[21] Frank Siqueira. “Especificação de Requisitos de Qualidade de Serviço na Análise e Projeto de Sistemas” Documento Interno, Departamento de Informática e Estatística, Universidade Federal de Santa Catarina. Florianópolis, SC, Brasil.

[22] Cristina Aurrecoechea, Andrew Campbell e Linda Hauw. “A Survey of Quality of Service Architectures”. In 4th IFIP International Conference on Quality of Service, Paris, France, March 1996.

[23] Svend Frolund e Jari Koistinen. “Quality of Service Specification in Distributed Object Systems Design”. In proceedings of the 4th USENIX Conference on Object Technologies and Systems (COOTS ‘98), Santa Fe, New Mexico, April 27-30, 1998.

[24] Svend Frolund e Jari Koistinen. “Quality of Service aware distributed object systems”, In proceedings of the 5th USENIX Conference on Oriented Technologies and Systems (COOTS '99) San Diego, California, USA, May 3-7,1999.

[25] John Zinky, David Bakken, Richard Schantz “Architectural Support for Quality of Service for CORBA Objects”, Theory and Practice of Object Systems, Vol. 3(1), January 1997.

[26] “Introduction to Quality of Service”, White Paper da Nortel Networks www.nortelnetworks.com.

[27] “Quality of Service Solutions”, White Paper da Cisco Systems. www.cisco.com.

[28] Gonçalo Quadros, Edmundo Monteiro, Fernando Boavida. “QoS em Sistemas de Comunicação: Desafios, Aproximações e Capacidades Desejáveis”, In Proceedings da I Conferencia Nacional de Telecomunicações, Aveiro, Abril de 1997.

[29] C.B. Montez, Um Modelo de Programação e uma Abordagem de Escalonamento Adaptativo Usando RT_CORBA, Tese de doutorado, Univ. Fed. de Santa Catarina, Departamento de Automação e Sistemas, Florianópolis, Brasil, 2000.

[30] A. Cervieri, Uma Abordagem de Escalonamento Adaptativo no Ambiente Real-Time CORBA, Univ. Fed. do Rio Grande do Sul, Instituto de Informática, Porto Alegre, Brasil, 2002.

[31] Jean Marie Farines, Joni S. Fraga e Rômulo De Oliveira, Sistemas de Tempo Real, p. 201, Escola de Computação 2000, São Paulo, Brasil

[32] C. Lu, J. A. Stankovic, T.F. Abdelzaher, G. Tao, S.H. Son, M. Marley. “Performance Specifications and Metrics for Adaptive Real-Time Systems”. 21° IEEE Real-Time Systems Symposium, Nov. 2000.

[33] C. Mcelhone, A. Burns. “Scheduling Optional Computations for Adaptive Real-Time Systems”. Journal of Systems Architectures 2000. http://www.researchindex.com.

[34] F. Siqueira. “Quartz: A QoS Architecture for Open Systems”, Department of Computer Science, Trinity College Dublin, December 1999.

[35] J. S. Fraga, R. J. Rabelo, F. A. Siqueira, J. Farines, C. B. Montez. “Suporte para Aplicações em Sistemas Abertos de Larga Escala: o Projeto SALE”. Dep. de Automação e Sistemas, Univ. Fed. de Santa Catarina, junho de 2001.

[36] J. Browne, P. Sackett, J. Wortmann, “Future Manufacturing Systems: Towards the Extended Enterprise”, Computer in Industry, Special Issue on CIM in the Extended Enterprise, Vol. 25(3), pp. 235-254, 1995.

[37] Object Managemet Group, “Realtime CORBA 1.0: Revised Submission”, OMG document orbos/98-12-10, Dezembro de 1998.

[38] OMG, “Security Service: v1.2 Final”, Object Management Group, Document 98-01-

02, in CORBAServices Nov. 1998.

[39] Object Management Group, “Fault-Tolerant CORBA”, Joint Revised Submission Document orbos/99-12-08, December 1999.

[40] Disponível em:
http://www.lfbs.rwth-aachen.de/users/stefan/rofes/docs/01-08-34.pdf

[41] Disponível em:
http://www.omg.org/technology/documents/vault.htm#corba_iiop

[42] C. Montez, J. Fraga, R. Oliveira. “Lidando com Sobrecarga no Escalonamento de Tarefas com Restrições Temporais: A Abordagem (p+i, k)-firm”, Anais do *VIII Simpósio de Computação Tolerante a Falhas*, Campinas, Brasil, 1999.

[43] M. Handaoui, P. Ramanathan. “ A Dynamic Priority Assignment Technique for Streams with Deadline (m,k)-firm”, *IEEE Trans. On Computer*, April 1995.

[44] J. Y. Chung. “ Scheduling Periodic Jobs that Allow Imprecise Results”, *IEEE Trans. On Computer*, 1990.

[45] K. G. Shin, C. L. Meissner. ”Adaptation and Graceful Degradation of Control System Performance by Task Reallocation and Period Adjustment”, *11° Euromicro Conference on Real-Time Systems*, jun. 1999.

[46] R. J. Rabelo “Interoperating Standards in Multiagent Manufacturing Scheduling Systems”, International Journal of Computer Applications in Technology (IJCAT), 2000.

[47] Disponível em:
http://www.corba.org/standards.htm

[48] Li, G. “Distributing Real-Time Objects : Some Early Experiences”, APM/ANSA External Paper 1231.01, Cambridge, UK, 1994.

[49] Frank Siqueira. “Especificação de Requisitos de Qualidade de Serviço em Sistemas Abertos: A Linguagem QSL”. 20º Simpósio Brasileiro de Redes de Computadores (SBRC'2002). Búzios - RJ, Brasil, Maio de 2002.

[50] Object Management Group, “Meta-Object Facility”, RFP5, Documento 96-05-02,1996.

[51] Object Management Group, “Portable Interceptor”, RFP - Request for Proposal OMG Document Orbos/ 98-05-05, Maio de 1998.

[52] C.L. Liu, J.W.Layland, “Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment”, Journal of the ACM, Vol. 20(1), January 1973.

[53] J.S.Fraga, J.-M. Farines, C. Montez; “Um Serviço de Tempo Global para Sistemas Distribuídos de Larga Escala”, 16° SBRC – Simpósio Brasileiro de Redes de Computadores, Rio de Janeiro-RJ, Maio de 1998.

[54] Maes, P. “Concepts and Experiments in Computational Reflection”, In OOPSLA, Orlando, Florida – USA, 1987.

[55] B. Sprunt, L. Sha, J. Lehoczky. “Aperiodic Task Scheduling for Hard-Real-Time Systems”. The Journal of Real-Time Systems, Vol. 1, 1989.

[56] D. L. Mills. “Internet Time Synchronization: The Network Time Protocol”, IEEE Trans. on Communications 39, October 1991.